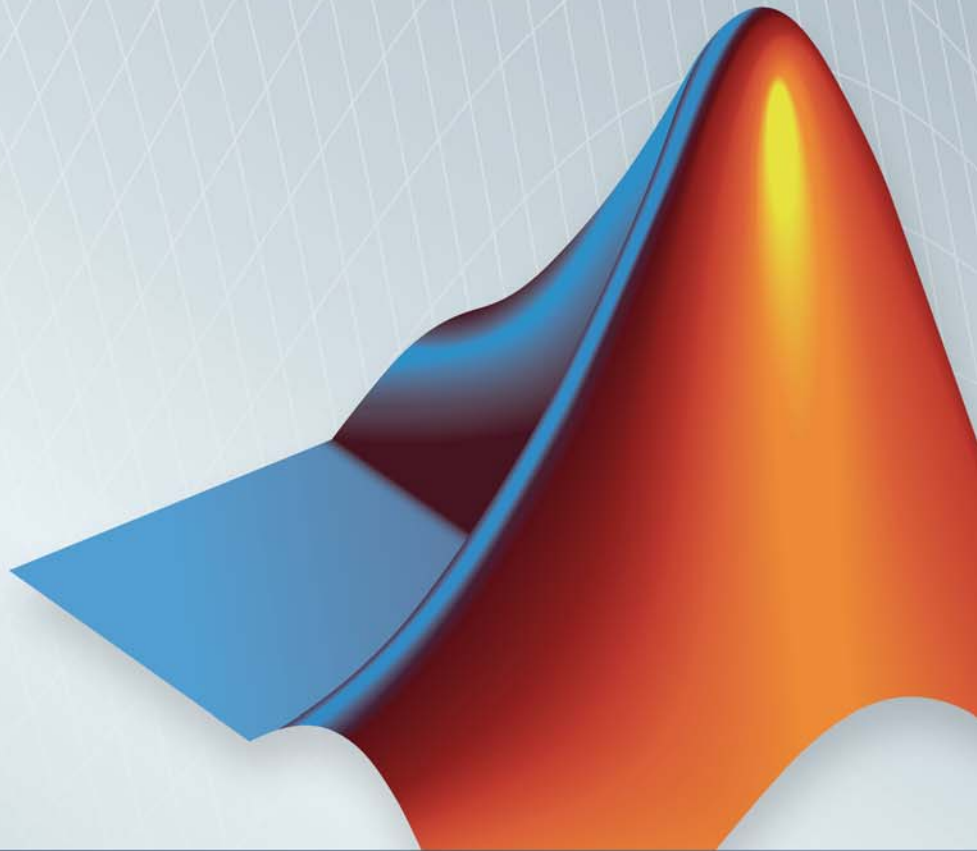


xPC Target™

API Guide

R2013a



MATLAB® & SIMULINK®



How to Contact MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

xPC Target™ API Guide

© COPYRIGHT 2002–2013 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)

Introduction

1

xPC Target APIs	1-2
xPC Target API for Microsoft .NET Framework	1-3
xPC Target C API	1-5
xPC Target COM API	1-7
Required Products	1-9

xPC Target API for Microsoft .NET Framework

2

Using the xPC Target API for .NET Framework	2-2
Features and Benefits	2-2
xpcosc Client Applications	2-3
File Server Browser Client Application	2-3
xPC Target .NET API Object Model	2-4
xPC Target API for .NET Framework Classes	2-5
Mathworks.xPCTarget.Framework.xPCTargetPC	2-5
Mathworks.xPCTarget.Framework.xPCApplication	2-6
Mathworks.xPCTarget.Framework.xPCScopes	2-6
Mathworks.xPCTarget.Framework.xPCParameters	2-6
Mathworks.xPCTarget.Framework.xPCParameter	2-6
Mathworks.xPCTarget.Framework.xPCSignals	2-7
Mathworks.xPCTarget.Framework.xPCSignal	2-7
Mathworks.xPCTarget.Framework.xPCAppLogger	2-7

xPC Target .NET API Usage	2-8
xPC Target .NET API Application Deployment	2-10

xPC Target API for C

3

Using the C API	3-2
Visual C Console Application	3-4
Target Application	3-4
Folders and Files	3-4
Building the xPC Target Application	3-6
Creating a Visual C Application	3-6
Building a Visual C Application	3-9
Running an xPC Target Visual C API Application	3-10
Using the xPC Target C API Application	3-10
C Code for sf_car_xpc.c	3-16

xPC Target API for COM

4

Using the COM API	4-2
Visual Basic GUI Using COM Objects	4-4
Target Application	4-5
Simulink Water Tank Model	4-5
Creating a Simulink Target Model	4-7
Tagging Block Parameters	4-8
Tagging Block Signals	4-12
Creating the Target Application and Model-Specific COM Library	4-14
Model-Specific COM Interface Library (model_nameCOMiface.dll)	4-17
Creating a New Microsoft Visual Basic Project	4-19

Referencing the xPC Target COM API and Model-Specific COM Libraries	4-21
Creating the Graphical Interface	4-23
Setting Properties	4-25
Writing Code	4-26
Creating the General Declarations	4-27
Creating the Load Procedure	4-27
Creating Event Procedures	4-28
Referencing Parameters and Signals Without Using Tags	4-34
Testing the Visual Basic Application	4-38
Building the Visual Basic Application	4-39
Deploying the API Application	4-39
Creating a New Visual Basic Project Using Microsoft Visual Studio 8.0	4-41

xPC Target API Examples

5

Visual Basic GUI Using .NET	5-2
Before Starting	5-2
Accessing the Demo Project Solution	5-3
Rebuilding the Demo Project Solution	5-3
Using the Demo Executable	5-4
 Visual Basic GUI Using COM	5-5
Before Starting	5-6
Accessing the sf_car_xpc Project	5-6
Rebuilding the sf_car_xpc Project	5-7
Using the sf_car_xpc Executable	5-7
 Command Line Scripts Using COM API	5-8
Tcl/Tk Scripts	5-8
Required Tcl/Tk Software	5-9
Using the Scripts	5-9

xPC Target API Reference for Microsoft .NET Framework

6

xPC Target API for Microsoft .NET Framework	
Classes	6-2
Target Computers	6-2
Target Applications	6-3
Scopes	6-3
Parameters	6-4
Signals	6-5
Data Logs	6-5
File Systems	6-6
Errors	6-6
xPC Target API for Microsoft .NET Framework — Alphabetical List	6-7

xPC Target API Reference for C

7

C API Functions		7-2
Target Computers		7-2
Target Applications		7-3
Scopes		7-4
Parameters		7-6
Signals		7-7
Data Logs		7-7
File Systems		7-8
Errors		7-9
C API Error Messages		7-10
C API Structures and Functions — Alphabetical List ..		7-14

8

COM API Methods	8-2
Target Computers	8-2
Target Applications	8-3
Scopes	8-4
Parameters	8-6
Signals	8-6
Data Logs	8-7
File Systems	8-7
Errors	8-8
COM API Methods — Alphabetical List	8-9

Index

Introduction

- “xPC Target APIs” on page 1-2
- “xPC Target API for Microsoft .NET Framework” on page 1-3
- “xPC Target C API” on page 1-5
- “xPC Target COM API” on page 1-7
- “Required Products” on page 1-9

xPC Target APIs

The xPC Target™ software provides several APIs that enable you to create custom applications to control real-time applications running on target computers.

The xPC Target software provides multiple types of xPC Target API (for example, the xPC Target API for Microsoft® .NET Framework, xPC Target C, and xPC Target COM). These interfaces provide the same functionality for you to write custom solutions (for example, client target applications and batch runs) that use the xPC Target software. The xPC Target documentation collectively refers to these APIs as xPC Target API.

The xPC Target APIs allow you to:

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

The following sections describe each library:

- “xPC Target API for Microsoft .NET Framework” on page 1-3
- “xPC Target C API” on page 1-5
- “xPC Target COM API” on page 1-7

xPC Target API for Microsoft .NET Framework

The xPC Target API for Microsoft .NET Framework consists of objects arranged in hierarchical order. Each of these objects has methods and properties that allow you to manipulate and interact with it. The API provides a number of classes, including those for target applications, scopes, the file system, and the target computer. The xPCTargetPC class is the main class that sits on top of a hierarchy of classes. This document presents the API reference. You can use these API functions from languages and applications that support managed code.

The Microsoft Windows® API supplies the infrastructure for using threads. The xPC Target 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.

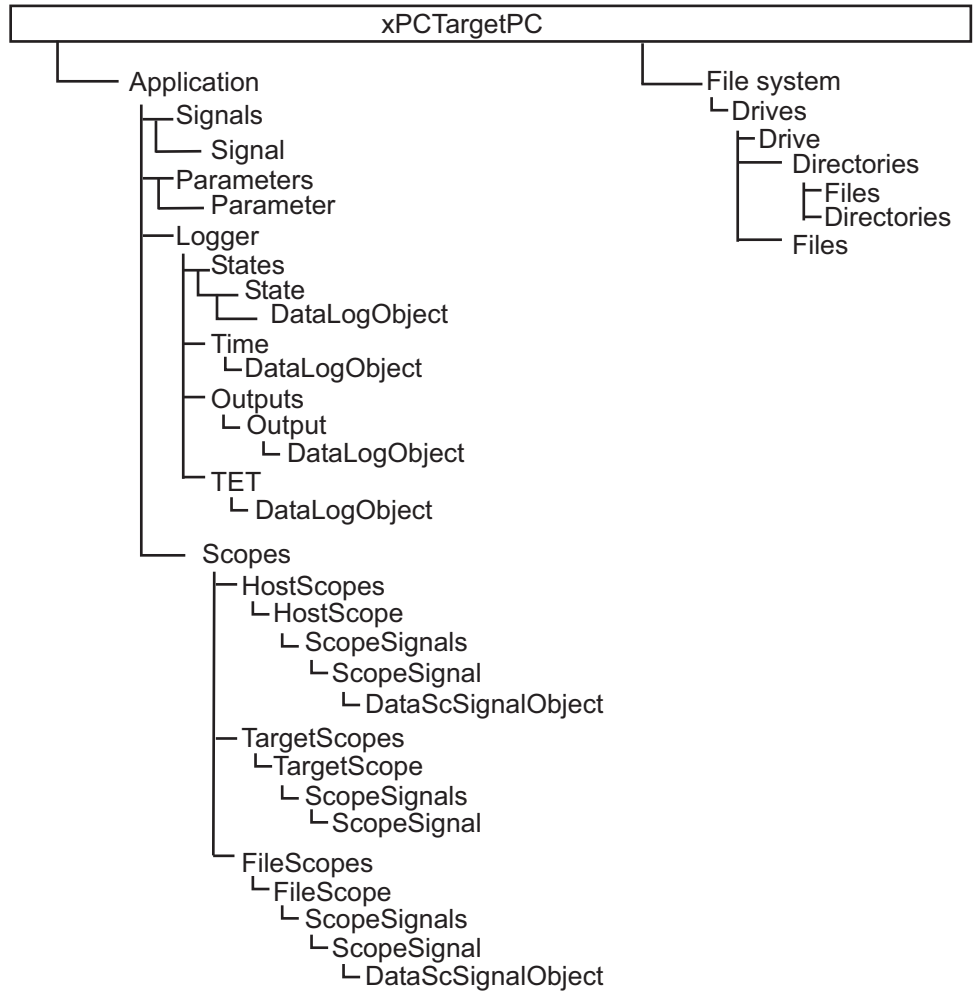
The xPC Target .NET object model closely models the xPC Target system. One xPCTargetPC Class object represents one xPC Target system.

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

An xPCFileSystem Class object represents the entire xPC Target file system. It contains objects like the following:

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

The following graphic outlines the xPCTargetPC hierarchy.



xPC Target C API

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

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

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

The xPC Target C API consists of blocking functions. For communications between the host and target computer, a default timeout of 5 seconds controls how long a target computer can take to communicate with a host computer.

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

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

xPC Target COM API

Note The xPC Target COM API is no longer being enhanced. You should use the xPC Target API for Microsoft .NET Framework instead. See “xPC Target API for Microsoft .NET Framework” on page 1-3

The xPC Target COM API is an open environment application program interface designed to work with Microsoft COM and the xPC Target C API. The COM API is not designed for multi-threaded operation.

With xPC Target COM API, you use a graphical development environment to create a GUI that works with an xPC Target application. Designed to work with Microsoft COM, the xPC Target COM API conforms to the component object model standard established by Microsoft.

The xPC Target COM API is a programming layer that sits between you and the xPC Target C API. The difference between the C API and this API is that while the C API is a dynamic link library of C functions, the xPC Target COM API dynamic link library is an organized collection of objects, classes, and functions. You access this collection through a graphical development environment such as Microsoft Visual Basic®. Using such a graphical development environment, you can create a custom GUI application that can work with one xPC Target application.

The xPC Target COM API library depends on `xpcapi.dll`, the xPC Target dynamic link library. However, the xPC Target C API is independent of the xPC Target COM API.

The xPC Target COM API consists of blocking functions. For communications between the host and target computer, a default timeout of 5 seconds controls how long a target computer can take to communicate with a host computer.

The xPC Target COM API has the following features:

- A DLL component server library — `xpcapicom.dll` is a component server DLL library COM interface consisting of component interfaces that access the target computer. The COM API library enhances the built-in

functionality of a programming language by allowing you to easily access the xPC Target C API for rapid development of xPC Target GUI.

- Built on top of the xPC Target C API — You can use the data, methods, and structured object model hierarchy in `xpcapicom.dll` to interface with an xPC Target application via an application such as Visual Basic. `xpcapicom.dll` also enables search functionality and bidirectional browsing capabilities. Generally, you view object models by selecting a type and viewing its members. Using the xPC Target COM API library, you can select a member and view the types to which it belongs.
- Programming language independent — This section describes how to create an xPC Target COM API application using Visual Basic. However, the xPC Target COM API interface is not limited to this third-party product. You can add the COM API library to development environments that can access COM libraries, such as Visual C++[®] or Java, as well as scripting languages such as Perl, Python, and Basic.
- Ideal for use with Visual Basic — The xPC Target COM API works well with Visual Basic, and extends the event-driven programming environment of Visual Basic.

Required Products

Refer to System Requirements for a list of the required xPC Target products. In addition, you need the following products:

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

xPC Target API for Microsoft .NET Framework

- “Using the xPC Target API for .NET Framework” on page 2-2
- “xPC Target .NET API Object Model” on page 2-4
- “xPC Target API for .NET Framework Classes” on page 2-5
- “xPC Target .NET API Usage” on page 2-8
- “xPC Target .NET API Application Deployment” on page 2-10

Using the xPC Target API for .NET Framework

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

In this section...
“Features and Benefits” on page 2-2
“xpcosc Client Applications” on page 2-3
“File Server Browser Client Application” on page 2-3

Features and Benefits

The xPC Target API for .NET framework includes the following features and benefits:

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

The xPC Target .NET API provides multiple ways for you to interface client side applications with target computers, including outside the MATLAB environment. For example

- Visual instrumentation for your real-time application
- Custom applications to perform data observation, collection, and archiving
- Real-time application debugging from a remote client computer
- Calibration, test, and evaluation of real-time processes
- Real-time data analysis

- Batch processing and automation scripts, which can run in a shell environment (such as PowerShell) or as a process console standalone application (.exe file)

xpcosc Client Applications

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

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

File Server Browser Client Application

The API xPC Target API for the .NET Framework has the following example, located in:

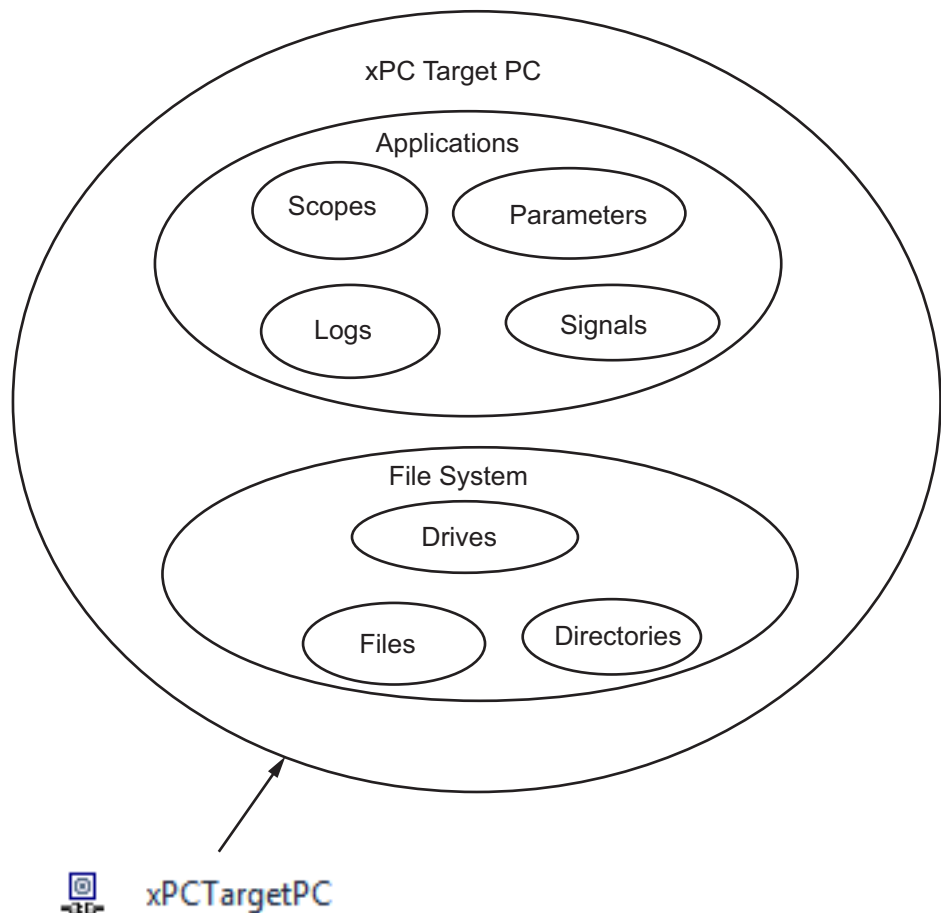
```
matlabroot\toolbox\rtw\targets\xpc\api\xPCFrameworkSamples\FileSystemBrowser
```

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

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

xPC Target .NET API Object Model

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



xPC Target API for .NET Framework Classes

The xPC Target .NET API provides an expansive object model layer. You should start your client model development on the following objects:

In this section...

“Mathworks.xPCTarget.Framework.xPCTargetPC” on page 2-5
“Mathworks.xPCTarget.Framework.xPCApplication” on page 2-6
“Mathworks.xPCTarget.Framework.xPCScopes” on page 2-6
“Mathworks.xPCTarget.Framework.xPCParameters” on page 2-6
“Mathworks.xPCTarget.Framework.xPCParameter” on page 2-6
“Mathworks.xPCTarget.Framework.xPCSignals” on page 2-7
“Mathworks.xPCTarget.Framework.xPCSignal” on page 2-7
“Mathworks.xPCTarget.Framework.xPCAppLogger” on page 2-7

Mathworks.xPCTarget.Framework.xPCTargetPC

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

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

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

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

Mathworks.xPCTarget.Framework.xPCApplication

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

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

Mathworks.xPCTarget.Framework.xPCScopes

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

Mathworks.xPCTarget.Framework.xPCParameters

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

Mathworks.xPCTarget.Framework.xPCParameter

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

Mathworks.xPCTarget.Framework.xPCSignals

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

Mathworks.xPCTarget.Framework.xPCSignal

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

Mathworks.xPCTarget.Framework.xPCAppLogger

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

xPC Target .NET API Usage

This topic presents the xPC Target API for .NET framework reference using the C# language and the Microsoft Visual Studio environment. At a minimum:

- Use the xPCTargetPC component in the Visual Studio environment. This addition provides convenient design-time features. To do this:
 - 1 Add the xPCTargetPC component to the Visual Studio Toolbox.
 - 2 To use this component, create a Windows application.
 - 3 Add an xPCTargetPC object to the application form by dragging an xPCTargetPC control from the Toolbox window to the design surface.

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

- Add a reference for xPCFramework.dll to your project (for example, to create a console application), include the following in your code. Doing so enables you to access the types available from the xPC Target environment using `MathWorks.xPCTarget.FrameWork`;
- To use the design-time capability of the Microsoft Visual Studio environment, copy the `xpcapi.dll` file to the same folder as the application executable. You also need this file to execute the application.

The xPC Target library has a 32-bit and a 64-bit version of the `xpcapi.dll`.

Note On 64-bit platforms, if you build a 64-bit target application in the Microsoft Visual Studio environment, and want to use the xPCTargetPC nonvisual component; place the 32-bit version of `xpcapi.dll` in the solution folder and place the 64-bit version of `xpcapi.dll` in the application folder that contains the `.exe` file. Placing the 32-bit version of `xpcapi.dll` in the solution folder enables you to use the design time capabilities of the Visual Studio environment.

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

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

xPC Target .NET API Application Deployment

This topic describes guidelines when distributing your xPC Target API for Microsoft .NET Framework GUI application:

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

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

xPC Target API for C

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

Using the C API

Keep the following guidelines in mind when you begin to write xPC Target C API applications with the xPC Target C API DLL:

- Carefully match the function data types as documented in the function reference. For C, the API includes a header file that matches the data types.
- To write a non-C application that calls functions in the xPC Target C API library, refer to the compiler documentation for a description of how to access functions from a library DLL. You must follow these directions to access the xPC Target C API DLL
- If you want to rebuild the model (`sf_car_xpc`), or otherwise use the MATLAB environment, you must have xPC Target Version 2.0 or later. To determine the version of xPC Target you are currently using, at the MATLAB command line, type

```
xpclib
```

This opens the xPC Target Simulink blocks library. The version of xPC Target should be at the bottom of the window.

- You can work with xPC Target applications with either MATLAB or an xPC Target C API application. If you are working with an xPC Target application simultaneously with a MATLAB session interacting with the target, keep in mind that only one application can access the target computer at a time. To move from the MATLAB session to your application, in the MATLAB Command Window, type

```
close(xpc)
```

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

- The xPC Target C API functions that communicate with the target computer check for timeouts during communication. If a timeout occurs, these functions will exit with the global variable `xPCError` set to either `ECOMTIMEOUT` (serial connections) or `ETCPTIMEOUT` (TCP/IP connections).

Use the `xPCGetLoadTimeOut` and `xPCSetLoadTimeOut` functions to get and set the timeout values, respectively.

There are a few things that are not covered in “C API Functions” and “C API Structures and Functions — Alphabetical List” for the individual functions, because they are common to almost all the functions in the xPC Target C API. These are

- Almost every function (except `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCGetLastError`, and `xPCErrorMsg`) has as one of its parameters the integer variable *port*. This variable is returned by `xPCOpenSerialPort` and `xPCOpenTcpIpPort`, and should be used to represent the communications link with the target computer.
- Almost every function (except `xPCGetLastError` and `xPCErrorMsg`) sets a global error value in case of error. The application obtains this value by calling the function `xPCGetLastError`, and retrieves a descriptive string about the error by using the function `xPCErrorMsg`. Although the actual error values are subject to change, a zero value typically means that the operation completed without producing an error, while a nonzero value typically signifies an error condition. Note also that the library resets the error value every time an API function is called; therefore, your application should check the error status as soon as possible after a function call.

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

Visual C Console Application

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

In this section...

“Target Application” on page 3-4

“Folders and Files” on page 3-4

“Building the xPC Target Application” on page 3-6

“Creating a Visual C Application” on page 3-6

“Building a Visual C Application” on page 3-9

“Running an xPC Target Visual C API Application” on page 3-10

“Using the xPC Target C API Application” on page 3-10

“C Code for sf_car_xpc.c” on page 3-16

Target Application

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

Folders and Files

This folder contains the C source of a Win32 console application that serves as an example for using the xPC Target C API. The `sf_car_xpc` files are in the folder

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

Filename	Description
VisualBasic\Models\ - sf_car_xpc\sf_car_xpc	Simulink model for use with xPC Target
VisualBasic\Models\ - sf_car_xpc\sf_car_xpc.dlm	Target application compiled from Simulink model
VisualC\sf_car_xpc.dsp	Project file for API application
sf_car_xpc.c	Source code for API application
VisualC\sf_car_xpc.exe	Compiled API application
VisualBasic\Models\ - xpcapi.dll	xPC Target C API functions for supported programming languages. Place this file in one of the following, in order of preference: <ul style="list-style-type: none"> • Folder from which the application is loaded • Windows system folder

The xPC Target C API files are in the folder

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

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

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

Building the xPC Target Application

These tutorials use the prebuilt xPC Target application

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

You can rebuild this application for your example:

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

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

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

```
sf_car_xpc2
```

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

Using Another C/C++ Compiler

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

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

Creating a Visual C Application

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

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

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

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

```
D:\mwd\sfc_car_xpc2
```

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

```
D:\mwd\sfc_car_xpc2
```

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

The Microsoft Visual C++ application is displayed.

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

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

- 6** In the left pane, select **C++ Source File**. In the right, enter the name of the file. For example, `sf_car_xpc.c`. Select the folder. For example, `C:\mwd\sfc_car_xpc2`.

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

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

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

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

- 11** In the left pane, select **Win32 Console Application**. On the right, enter the name of the project. For example, `sf_car_xpc`. Select the working folder from step 1. For example, `C:\mwd\sfc_car_xpc2`.

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

A Win32 Console Application dialog is displayed.

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

14 Click **Finish**.

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

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

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

```
D:\mwd\sfc_car_xpc2\sfc_car_xpc.c
```

Click **OK**.

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

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

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

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

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

Placing the Target Application File in a Different Folder

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

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

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

Building a Visual C Application

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

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

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

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

When you are ready to run your Visual C Application, go to "Running an xPC Target Visual C API Application" on page 3-10.

Running an xPC Target Visual C API Application

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

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

Using the xPC Target C API Application

To run a xPC Target C API application, you must have a working target computer running at least xPC Target Version 2.0 (Release 13).

This tutorial assumes that you are using the xPC Target C API application `sf_car_xpc.exe` that comes with xPC Target. In turn, `sf_car_xpc.exe` expects that the xPC Target application is `sf_car_xpc.dlm`.

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

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

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

How to Run the `sf_car_xpc` Executable

- 1 Create an xPC Target boot disk with a serial or network communication. If you use serial communications, set the baud rate to 115200. Otherwise, create the boot disk as directed in xPC Target Getting Started.
- 2 Start the target computer with the xPC Target boot disk.

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

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

or

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

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

```
close(xpc)
```

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

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

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

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

```
D:\mwd\sfc_car_xpc2
```

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

The syntax for the example command is

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

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

```
sf_car_xpc -t 192.168.0.1:2222
```

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

```
sf_car_xpc -c 0
```

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

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

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

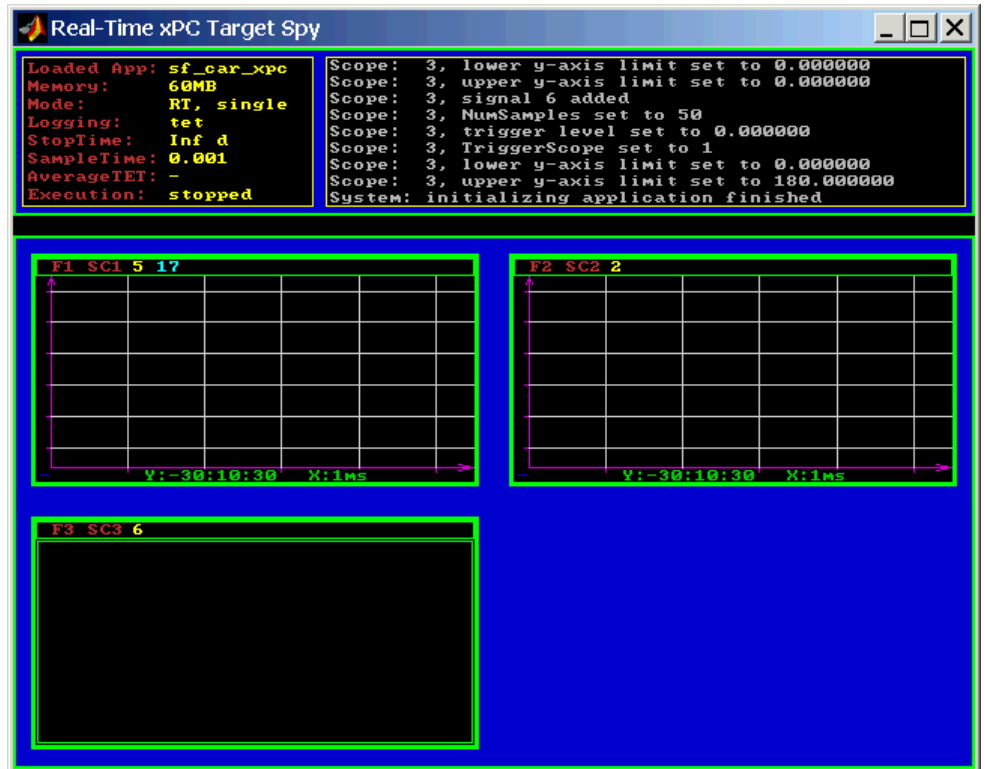
R	The status of the target application: R if running, N if stopped
Br	The brake torque; legal values range from 0 to 4000
Th	The throttle as a percentage (0 - 100) of the total
G	Gear the vehicle is in (ranges between 1 and 4)
VehSpeed	Speed of the vehicle in miles per hour
VehRPM	Revolutions per minute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

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

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

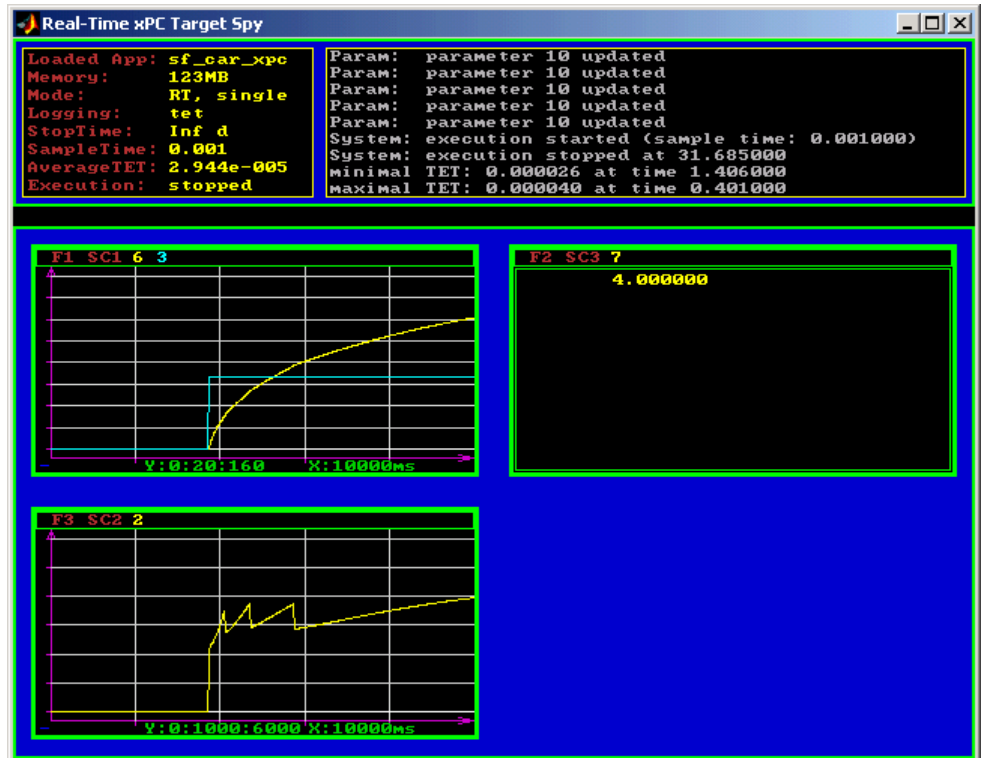
Note Note that a positive value for the brake automatically sets the throttle value to 0, and a positive value for the throttle automatically sets the brake value to 0.

The target computer displays the following messages and three scopes.



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

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



The first scope (SC1) shows the throttle rising to a maximum value of 100 and the vehicle speed gradually increasing. The third scope (SC3) shows the vehicle RPM. Notice the changes in the vehicle RPM as the gears shift from first to fourth gear as displayed in the third numerical scope (SC2).

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

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

C Code for sf_car_xpc.c

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

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

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

/* xPC Target C-API specific includes */
#include "xpcapi.h"
#include "xpcapiconst.h"

#define SERIAL 0
#define TCPIP 1

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

/* Global Variables */
int mode = TCPIP, comPort = 0;
```

```

int    port;
int    thrPID, brakePID, rpmSID, speedSID, gearSID;
char *ipAddress, *ipPort, *pathToApp = NULL;

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

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

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

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

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

```

```

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

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

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

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

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

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

    throttle = getParam(thrPID);
    brake    = getParam(brakePID);
    fputs("\nR   Br   Th  G   VehSpeed   VehRPM  \n", stdout);
    fputs("  -   - - - -  - -  -   - - - - - - - - - - - - - - - \n", stdout);
    while (1) {
        if (_kbhit()) {

```



```
c = _getch();
switch (c) {
    case 't':
        if (throttle)
            setParam(thrPID, --throttle);
        break;
    case 'T':
        if (brake)
            setParam(brakePID, (brake = 0));
        if (throttle < 100)
            setParam(thrPID, ++throttle);
        break;
    case 'b':
        setParam(brakePID, (brake = max(brake - 200, 0)));
        if (brake)
            setParam(thrPID, (throttle = 0));
        break;
    case 'B':
        if (throttle)
            setParam(thrPID, (throttle = 0));
        setParam(brakePID, (brake = min(brake + 200, 4000)));
        break;
    case 's':
    case 'S':
        if (xPCIsAppRunning(port)) {
            xPCStopApp(port); checkError(NULL);
        } else {
            xPCStartApp(port); checkError(NULL);
        }
        break;
    case 'q':
    case 'Q':
        return;
        break;
    default:
        fputc(7, stderr);
        break;
}
} else {
    Sleep(50);
```

```

    }
    printf( "\r%c  %4d %3d %1d %10.3f %10.3f",
            (xPCIsAppRunning(port) ? 'Y' : 'N'),
            (int)brake, (int)throttle,
            (int)xPCGetSignal(port, gearSID),
            xPCGetSignal(port, speedSID),
            xPCGetSignal(port, rpmSID));
    }
} /* end processKeys() */

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

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

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

```

```

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

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

```

```

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

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

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

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

```

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

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

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

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


xPC Target API for COM

- “Using the COM API” on page 4-2
- “Visual Basic GUI Using COM Objects” on page 4-4

Using the COM API

Note The xPC Target COM API is no longer being enhanced. You should use the xPC Target API for Microsoft .NET Framework instead. See “xPC Target API for Microsoft .NET Framework” on page 1-3.

This topic describes how to write a custom application using the xPC Target COM API. This COM API enables you to write COM applications to load, run, and control an xPC Target application.

Before you start, read this section for guidelines on writing custom applications based on the xPC Target COM API. You do not need to be a seasoned C or C++ programmer to follow these procedures or to write custom applications with the xPC Target COM API. You should, however, have some rudimentary programming knowledge. For this topic, you will be using Microsoft Visual Basic.

This example uses the model `xpctank`. If you want to rebuild this model, or otherwise use the MATLAB software, you must have xPC Target software version 2.0 or higher.

To determine which version of the software you are currently using, at the MATLAB command line, type

```
xpcLib
```

This opens the xPC Target Simulink blocks library. The xPC Target software version of should be at the bottom of the window.

You can work with xPC Target applications with either the MATLAB software or an xPC Target COM API application. If you are working with an xPC Target application using an xPC Target COM API application simultaneously with a MATLAB session interacting with the target, keep in mind that only one application can access the target computer at a time. To move from the MATLAB session to your application, in the MATLAB Command Window, type

```
close(xpc)
```


This frees the connection to the target computer for use by your xPC Target COM API application. Conversely, you will need to have your COM API application call the `Close` method to enable access to the target from a MATLAB session.

Although you are building an xPC Target COM API application, you still need to access the `xpcapi.dll`. When distributing the xPC Target COM API application, place this file in one of the following, in order of preference:

- Folder from which application is loaded
- Windows system folder

To create the target application and build associated COM objects from the tagged signals and parameters, use a supported Microsoft Visual C++ compiler to generate code.

Note You can use an Express edition of a compiler to reference the COM API and generated signals and parameters when creating models. However, you cannot use an Express edition of a compiler to generate code for these COM objects.

Visual Basic GUI Using COM Objects

In this section...
“Target Application” on page 4-5
“Simulink Water Tank Model” on page 4-5
“Creating a Simulink Target Model” on page 4-7
“Tagging Block Parameters” on page 4-8
“Tagging Block Signals” on page 4-12
“Creating the Target Application and Model-Specific COM Library” on page 4-14
“Model-Specific COM Interface Library (model_nameCOMiface.dll)” on page 4-17
“Creating a New Microsoft® Visual Basic® Project” on page 4-19
“Referencing the xPC Target COM API and Model-Specific COM Libraries” on page 4-21
“Creating the Graphical Interface” on page 4-23
“Setting Properties” on page 4-25
“Writing Code” on page 4-26
“Creating the General Declarations” on page 4-27
“Creating the Load Procedure” on page 4-27
“Creating Event Procedures” on page 4-28
“Referencing Parameters and Signals Without Using Tags” on page 4-34
“Testing the Visual Basic Application” on page 4-38
“Building the Visual Basic Application” on page 4-39
“Deploying the API Application” on page 4-39
“Creating a New Visual Basic Project Using Microsoft® Visual Studio® 8.0” on page 4-41

Target Application

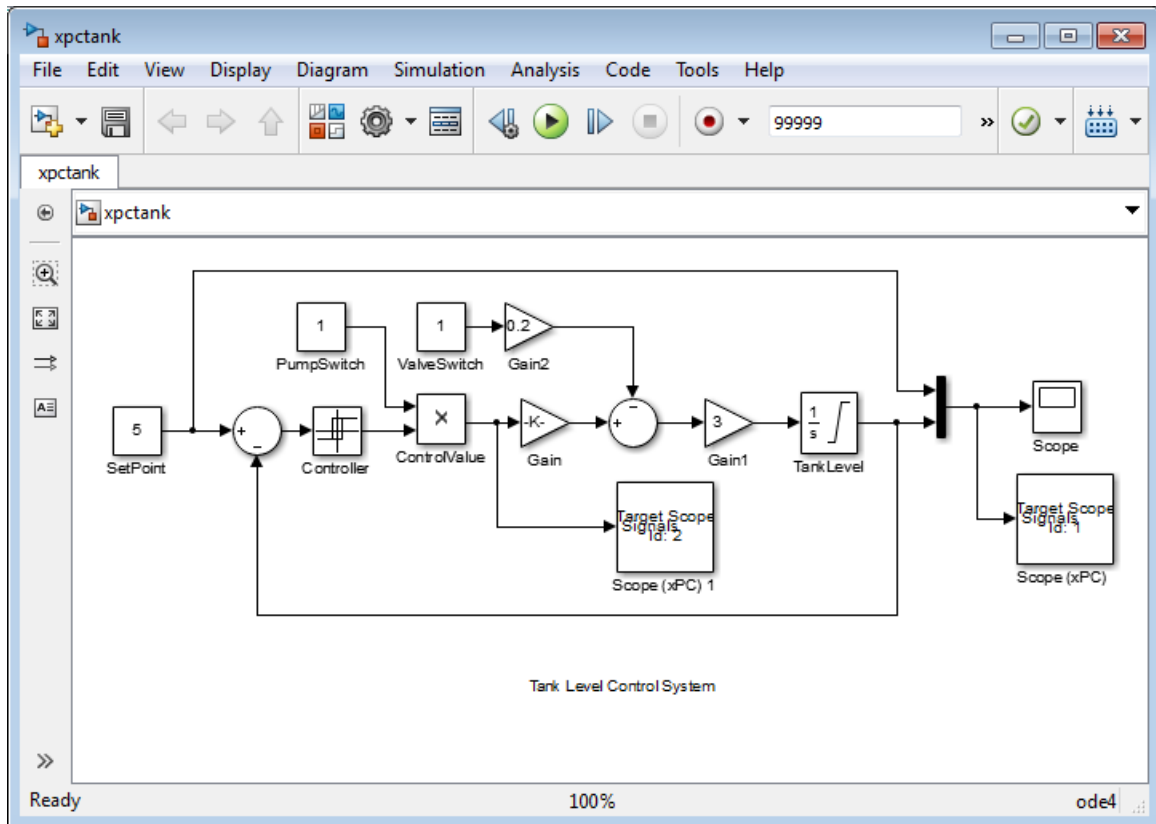
Note

- The xPC Target COM API is no longer being enhanced. You should use the xPC Target API for Microsoft .NET Framework instead. See “xPC Target API for Microsoft .NET Framework” on page 1-3.
 - These topics assume that you know how to create projects and forms in Microsoft Visual Basic, and that you are familiar with the concept of automatic code completion. For further details on Microsoft Visual Basic, refer to your Microsoft product documentation.
-

These topics use the Simulink model `xpctank` and requests that you enter tags for signals and parameters to create the Simulink model `xpc_tank1`. You will then build the real-time target application `xpc_tank1.dlm` and the GUI `xpc_tank1_COM.exe` application using the xPC Target COM API library and Microsoft Visual Basic.

Simulink Water Tank Model

The xPC Target software includes the Simulink model `xpctank`. This is a model of a water tank with a pump, drain, and valve controller.



TankLevel — The water level in the tank is modeled using a limited integrator named TankLevel.

PumpSwitch — The pump can be turned off manually to override the action of the controller. This is done by setting PumpSwitch to 0. When PumpSwitch is 1, the controller can use the control valve to pump water into the tank.

ValveSwitch (drain valve) — The tank has a drain valve that allows water to flow out of the tank. Think of this as water usage or consumption that reduces the water level. This behavior is modeled with the constant block named ValveSwitch, the gain block Gain2, and a summing junction. The minus sign on the summing junction produces a negative flow rate (drain), which reduces the water level in the tank.

When ValveSwitch is 0 (closed), the valve is closed and water cannot flow out of the tank. When ValveSwitch is 1 (open), the valve is open and the water level is reduced by draining the tank.

Controller — The controller is very simple. It is a bang-bang controller and can only maintain the selected water level by turning the control valve (pump valve) on or off. A water level set point defines the desired median water level. Hysteresis enables the pump to avoid high-frequency on and off cycling. This is done using symmetric upper and lower bounds that are offsets from the median set point. As a result, the controller turns the control valve (pump valve) on whenever the water level is below the set point minus the offset. The summing junction compares this lower bound against the tank water level to determine whether or not to open the control valve. If the pump is turned on (PumpSwitch is 1) water is pumped into the tank. When the water level reaches or exceeds the set point plus the upper bound, the controller turns off the control valve. When the water level reaches this boundary, water stops pumping into the tank.

Scope blocks — A standard Simulink Scope block is added to the model for you to view signals during a simulation. xPC Target Scope blocks are added to the model for you to view signals while running the target application. Scope id:1 displays the actual water level and the selected water level in the tank. Scope id:2 displays the control signals. Both scopes are displayed on the target computer using a target scope.

The xpctank model is built entirely from standard Simulink blocks and scope blocks from the xPC Target software. It does not differ from a model you would normally use with the software.

Creating a Simulink Target Model

A target application model is a Simulink model that describes your physical system and its behavior. You use this model to create a real-time target application, and you use this model to select the parameters and signals you want to connect to a custom graphical interface.

You do not have to modify this model when you use it with Simulink 3D Animation™ or other third-party graphical elements.

Create a target application model before you tag block parameters and block signals to create a custom graphical interface:

- 1 In the MATLAB Command Window, type

```
xpctank
```

A Simulink model for a water tank opens. This model contains a set of equations that describe the behavior of a water tank and a simple controller.

The controller regulates the water level in the tank. This model contains only standard Simulink blocks and you use it to create the xPC Target application.

- 2 From the **File** menu, click **Save as** and enter a new filename. For example, enter `xpc_tank1` and then click **OK**.

Note If you save your own copy of `xpctank`, be sure to be in the folder that contains that model before calling it from the MATLAB window.

Your next task is to mark the block properties and block signals. See “Tagging Block Parameters” on page 4-8 and “Tagging Block Signals” on page 4-12. Building an xPC Target application that has been tagged generates a model-specific COM library, `model_nameifaceCOM.dll`, which you can later reference when writing your xPC Target COM API application.

Tagging Block Parameters

Tagging parameters in your Simulink model enables you to generate a model-specific COM library to provide access to model parameter IDs via the xPC Target COM API library. These interface blocks contain the parameters you connect to control devices (such as sliders) in your model. Tagging parameters makes it easier for you to refer to these parameters later, when you write your xPC Target COM API application.

Note If you do not tag parameters before you generate your Simulink model, you must specify model parameters manually. See “Referencing Parameters and Signals Without Using Tags” on page 4-34 for this procedure.

This procedure uses the model `xpc_tank1` (or `xpctank`) as an example. See “Creating a Simulink Target Model” on page 4-7.

Tip The `xpctank` model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

- 1 Open a Simulink model. For example, in the MATLAB window type `xpc_tank1` or `xpctank`.
- 2 Point to a Simulink block, and then right-click. For example, right-click the `SetPoint` block.
- 3 From the menu, click **Properties**.

A block properties dialog box opens.

- 4 In the **Description** box, delete the existing tag and enter a tag to the parameters for this block.

For example, the `SetPoint` block is a constant with a single parameter that selects the level of water in the tank. Enter the tag:

```
xPCTag(1)=water_level;
```

Tags have the following format:

```
xPCTag(1, . . . index_n)= label_1 . . . label_n;
```

- `index_n` — Index of a block parameter. Begin numbering parameters with an index of 1.

- `label_n` — Name for a block parameter to connect to a property for the parameter you tag in the model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

You can assign multiple labels to one tag, such as

```
xPCTag(1)=label;xPCTag(1)=label2;
```

You might want to assign multiple labels if you want to tag a parameter for different purposes. For example, you can tag a parameter to create a model-specific COM library. You might also want to tag a parameter to enable the function `xpcsliface` to generate a user interface template model.

You can also issue one tag definition per line, such as

```
xPCTag(1)=label;  
xPCTag(2)=label2;
```

- 5** Repeat step 4 for the remaining parameters you want to tag.

For example, for the Controller block, enter the tags:

```
xPCTag(1,2,3)=upper_water_level lower_water_level  
pump_flowrate;
```

For the PumpSwitch and ValveSwitch blocks, enter the tags:

```
xPCTag(1)=pump_switch;
```

and

```
xPCTag(1)=drain_valve;
```

To tag a block with four properties, use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```


To tag a block with at least four properties for the second and fourth properties, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

- 6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpc_tank1
```

Your next task is to tag block signals, if you have not already done so; then, create the model. See “Tagging Block Signals” on page 4-12.

Tagging Block Signals

Tagging signals in your Simulink model enables you to generate a model-specific COM library to provide access to model signal IDs via the COM API library. These interface blocks contain the signals you connect to display devices (such as labels) in your model. Tagging signals makes it easier for you to refer to these signals later, when you write your xPC Target COM API application. After you tag signals, you will be ready to build your xPC Target application.

Note If you do not tag signals before you generate your Simulink model, you must specify model signals manually. See “Referencing Parameters and Signals Without Using Tags” on page 4-34 for this procedure.

This procedure uses the model `xpc_tank1` (or `xpctank`) as an example. See “Creating a Simulink Target Model” on page 4-7.

Tip The `xpctank` model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

Notice that you cannot select signals on the output ports of virtual blocks, such as Subsystem and Mux blocks. Also, you cannot select signals on function call signal output ports.

- 1 Open a Simulink model. For example, in the MATLAB window type `xpc_tank1` or `xpctank`.

- 2 Point to a Simulink signal line, and then right-click.
- 3 From the menu, click **Signal Properties**. For example, right-click the signal line from the TankLevel block.

A Signal Properties dialog box opens.

- 4 Select the **Documentation** tab.
- 5 In the **Description** box, enter a tag to the signals for this line. For example, the TankLevel block is an integrator with a single signal that indicates the level of water in the tank. Enter the tag:

```
xPCTag(1)=water_level;
```

- 6 Repeat step 5 for the remaining signals you want to tag. For example, for the signal from the ControlValve block, enter the tag:

```
xPCTag=pump_valve;
```

Signal tags have the following syntax:

```
xPCTag(1, . . . index_n)=label_1 . . . label_n;
```

- `index_n` — Index of a signal within a vector signal line. Begin numbering signals with an index of 1.
- `label_n` — Name for a signal to connect to a property for the signal you tag in the model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

For single-dimension ports, the following syntax is also valid:

```
XPCTag=label;
```

You can assign multiple labels to one tag, such as

```
xPCTag(1)=label1;xPCTag(1)=label2;
```

You might want to assign multiple labels if you want to tag a signal for different purposes. For example, you can tag a signal to create a model-specific COM library. You might also want to tag a signal to enable the function `xpcsliface` to generate a user interface template model.

You can also issue one tag definition per line, such as

```
xPCTag(1)=label1;  
xPCTag(2)=label2;
```

To tag a signal line with four signals (port dimension of 4) use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```

To tag the second and fourth signals in a signal line with at least four signals, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

- 7 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpc_tank1
```

Create the target application. See “Creating the Target Application and Model-Specific COM Library” on page 4-14.

Creating the Target Application and Model-Specific COM Library

Use this procedure to create a target application that you want to connect to a GUI application and the model-specific COM interface library (`model_nameCOMiface.dll`).

After you copy a Simulink model and tag the block parameters and block signals, you can create a target application and download it to the target

computer. This procedure uses the Simulink model `xpc_tank1` (or `xpctank`) as an example (see “Creating a Simulink Target Model” on page 4-7).

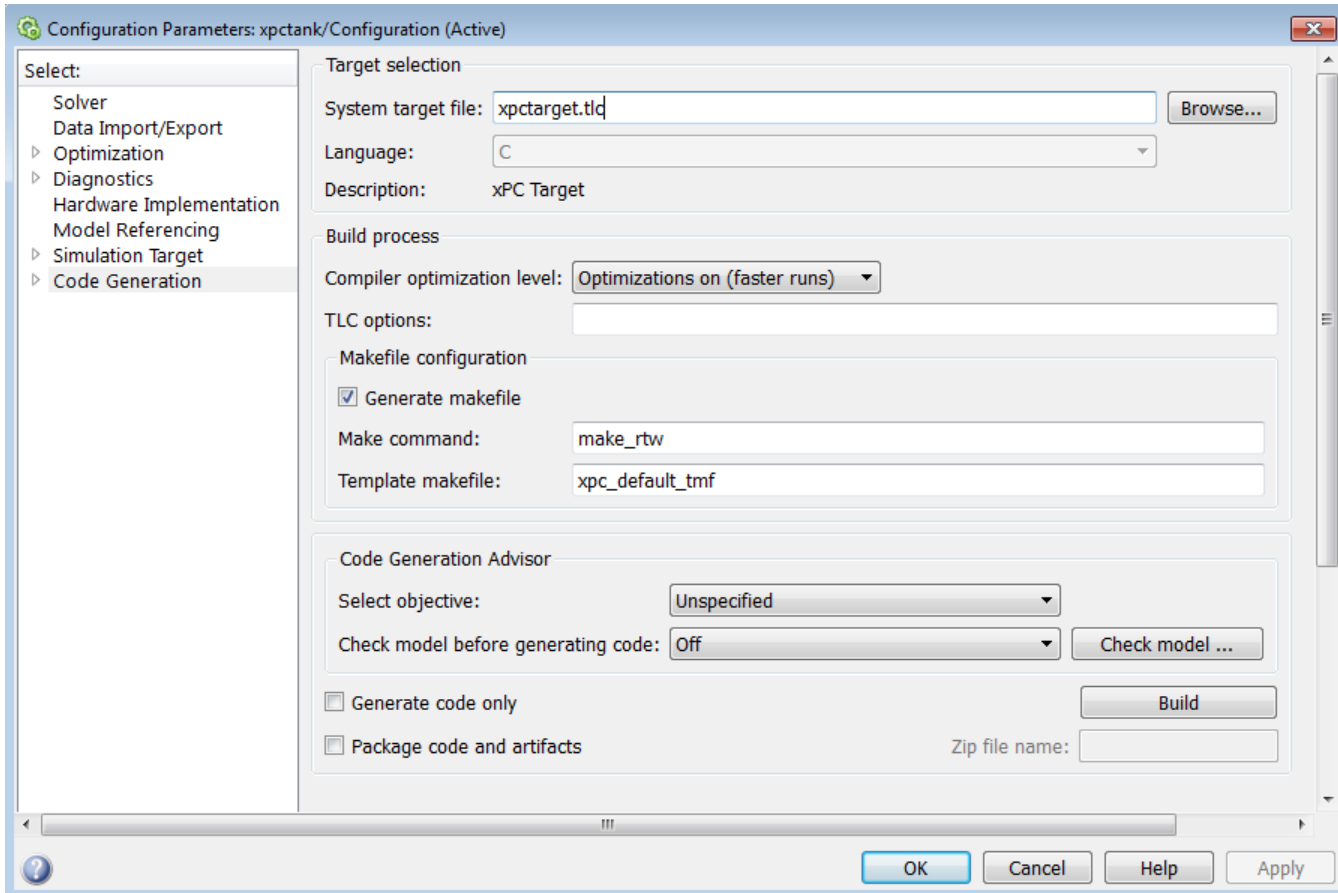
- 1** If this is a new release of the product, configure the host computer with the required settings, including the compiler.
- 2** Start or reset the target computer with the boot media (CD, DVD, or network image). Close other applications loaded on the target computer.
- 3** In the MATLAB window, type `xpc_tank1` or `xpctank`.

A Simulink window opens with the model file.

- 4** From the **Simulation** menu, click **Model Configuration Parameters**.

The Configuration Parameters dialog is displayed for the model.

- 5** In the left pane, click the **Code Generation** node.
- 6** In the **Target selection** section, click the **Browse** button at the **System target file** list. Click `xpctarget.tlc`. The Code Generation pane looks like this:



Click **OK**.

- 7 In the left pane, click the **xPC Target options** node.

The **xPC Target options** pane is displayed. Verify that the options **Automatically download application after building** and **Download to default target PC** are set.

- 8 Click the **Solver** node.

The **Solver** pane is displayed.

- 9 Check that the **Stop time** is long enough for you to interact with the target application.
- 10 Click **OK** to save and exit.
- 11 From the **Code** menu, click **C/C++ Code > Build Model**.

The Simulink Coder, xPC Target, and a third-party C compiler create the target application `xpc_tank1.dlm` and the COM object library `xpc_tank1COMiface.dll`. The target application is also downloaded to the target computer.

- 12 If you want, you can close the MATLAB Command Window.

Your next task is to create a Microsoft Visual Basic API application using COM objects. This API application connects and controls the target application. See “Creating a New Microsoft® Visual Basic® Project” on page 4-19. For more information about model-specific COM interface library, refer to “Model-Specific COM Interface Library (model_nameCOMiface.dll)” on page 4-17.

Model-Specific COM Interface Library (model_nameCOMiface.dll)

The generated model-specific COM interface library is a DLL component server library that enhances programming using the xPC Target COM API library. A model-specific COM interface library is specific to the model from which it is generated; do not reference a model-specific library for another model. If you choose not to generate a model-specific COM interface library, refer to “Referencing Parameters and Signals Without Using Tags” on page 4-34 for a description of how to otherwise reference parameters and signals in the xPC Target COM API application.

The mode-specific COM interface library allows users easy access to preselected tagged signals and desired tagged parameters for use in conjunction with the xPC Target COM API `xPC Target` and `xPCScope` object signal monitoring and parameter member functions such as `xPCTarget.GetParam`, `xPCTarget.SetParam`, and `xPCTarget.GetSignal`.

The xPC Target COM generated objects are of two types:

- `model_namebio`
- `model_namept`

where `model_name` is the name of the Simulink model. The `model_namebio` type is for tagged block I/O signals and the `model_namept` type is for tagged parameters.

Model-Specific COM Signal Object Classes

Model-specific COM signal classes have two types of members in which you are interested, the `Init` function and class properties. You will find these members in the `model_namebio` class, where `model_name` is the name of your model.

The `Init` function invokes the `Init` method once, passing it the `Ref` property from the `xPCProtocol` class. This method initializes the object to communicate with the target computer to access the signal identifiers when accessing the object's properties. Refer to the call in the Microsoft Visual Basic code example in “Creating the Load Procedure” on page 4-27.

Each class has a list of properties (specified in the `Tag` syntax in the **Description** field of the signal property). These properties return the xPC Target signal identifiers or signal numbers of the tagged signals. The generated property name is the name specified in the tagged signal description using the following syntax:

```
xPCTag=Property name;
```

For example, in the model `xpc_tank1`, there are two signal tags in the **Description** field:

- The output from the integrator block labeled `TankLevel` is tagged `xPCTag=water_level`.
- The output from the multiply block labeled `ControlValve` is tagged `xPCTag=pump_valve`.

Model-Specific COM Parameter Object Classes

Model-specific COM signal classes have two types of members in which you are interested, the `Init` function and class properties. You will find these

members in the `model_namept` class, where `model_name` is the name of your model.

The `Init` function invokes the `Init` method once, passing it as input the `Ref` property from the `xPCProtocol` class. This method initializes the object to communicate with the target computer to access the parameter identifiers when accessing the object's properties. Refer to the call in the Microsoft Visual Basic code example in “Creating the Load Procedure” on page 4-27.

Each class has a list of properties (specified in the `Tag` syntax in the **Description** field of the block property). These properties return the xPC Target parameter identifiers of the tagged parameters. The generated property name is the name specified in the tagged signal description using the following syntax:

```
xPCTag(1)=Property name;
```

For example, in the model `xpc_tank1`, there are two parameter tags in the **Description** field:

- The parameter for SetPoint blocks is tagged `xPCTag=set_water_level;`
- The parameters for the Controller block are tagged
`xPCTag(1,2,3,)=upper_water_level lower_water_level
pump_flowrate;`

Creating a New Microsoft Visual Basic Project

The following procedures describe how you can create a Microsoft Visual Basic project to take advantage of the xPC Target COM API to create a custom GUI for the xPC Target application. The procedures build on the `xpctank` (`xpc_tank1`) model you saved earlier (see “Creating the Target Application and Model-Specific COM Library” on page 4-14). The Microsoft Visual Basic environment allows you to interact with your target application using a GUI while the target application is running in real time on the target computer.

The procedures for the following topics apply to Microsoft Visual Studio 6.0. To use Microsoft Visual Studio 8.0 instead, see “Creating a New Visual Basic Project Using Microsoft® Visual Studio® 8.0” on page 4-41.

- 1 Create a new project folder.

From the folder `matlabroot\toolbox\rtw\targets\xpc\api`, copy the file `xpcapi.dll` (API library) to this new project folder. Alternatively, you can copy the file `xpcapi.dll` into the Windows system folder.

You do not need to copy `xpcapiCOM.dll` (the COM API library) into the current folder, but register it in your system (see “Registering Dependent Dynamic Link Libraries” on page 4-40.)

- 2 From your MATLAB working folder, copy the files `model_name.dlm` (target application) and `model_nameCOMiface.dll` (model-specific COM library) to the new project folder.
- 3 While in this project folder, open Microsoft Visual Basic. From the **File** menu, click **New Project**.

The New Project dialog box opens.

Note

- Microsoft product screen shots reprinted with permission from Microsoft Corporation.
 - Be sure to open the Microsoft Visual Basic project from the project folder itself, not from Microsoft Visual Basic.
-

- 4 Select **Standard EXE**, and then click **OK**.

The Microsoft Visual Basic Integrated Development Environment opens with a blank form.

- 5 From the **File** menu, click **Save Project As** and enter a filename for the form and the project. For example, for the form, enter

```
xpc_tank1_COM.frm
```

At the project prompt, enter

```
xpc_tank1_COM.vbp
```

Referencing the xPC Target COM API and Model-Specific COM Libraries

You need to reference the xPC Target COM API and model-specific COM libraries so that Microsoft Visual Basic will use them in the current project. Assuming that you created the Visual Basic project as described in the preceding procedure, reference the library as described in this procedure:

- 1 From the **Project** menu, click **References**.

The References dialog box opens.

- 2 Select the **COM** tab.

- 3 Scroll down the **Component Name** list to the bottom. Select the **xPC Target API COM Type Library** check box.

- 4 Click **Select**.

- 5 Click **OK**.

The xPC Target COM API Type library (xpcapiCOM.dll) is now available for use in your project.

- 6 To add the model-specific COM library, click **References** again from the **Project** menu.

The References dialog box opens.

- 7 Scroll to find your model name. Select the check box **xpc_tank1COMiface 1.0 Type Library**.

- 8 Click **Select**.

- 9 Click **OK**.

The model-specific COM API Type Library (xpc_tank1COMiface.dll) is now available for use in your project. Sections “Viewing Model-Specific COM Signal Object Classes” on page 4-22 and “Viewing Model-Specific COM Parameter Object Classes” on page 4-22 describe how to look at class objects.

Because the xPC Target COM API is an add-on to Visual Basic, it might help to know a bit about Visual Basic before going much farther with using the COM API. The section “Creating the Graphical Interface” on page 4-23 shows you how to use Visual Basic to create a project for the `xpctank` or (`xpc_tank1`) model.

Viewing Model-Specific COM Signal Object Classes

After you create a Visual Basic project and reference the xPC Target COM API and model-specific COM libraries, you can use the Visual Basic Object browser (click the **View** menu and select **Object Browser**) to look at the objects for the `xpctankbio` or `xpc_tank1bio` class:

- 1 From the **View** menu, select **Object Browser**.

A dialog box pops up with a drop-down list containing the type library information for a project.

- 2 Select the drop-down list for the project/library.

A list of the project libraries appears.

- 3 Select `model_nameCOMIFACELib`.

The classes in your model appear.

- 4 To view the objects of a class, select that class.

The objects in your class appear.

The `xpctankbio` (or `xpc_tank1bio`) class contains the function `Init` and the two properties

- `water_level`
- `pump_valve`

Viewing Model-Specific COM Parameter Object Classes

After you create a Visual Basic project and reference the xPC Target COM API and model-specific COM libraries, you can use the Visual Basic Object

browser (click the **View** menu and select **Object Browser**) to look at the objects for the `xpctankpt` or `xpc_tank1pt` class:

1 From the **View** menu, select **Object Browser**.

A dialog box pops up with a drop-down list containing the type library information for a project.

2 Select the drop-down list for the project/library.

A list of the project libraries appears.

3 Select `model_nameCOMIFACELib`.

The classes in your model appear.

4 To view the objects of a class, select that class.

The objects in your class appear.

The `xpctankpt` (or `xpc_tank1pt`) class contains the method `Init` and the member properties





- `pump_switch`
- `upper_water_level`
- `lower_water_level`
- `pump_flowrate`
- `water_level`
- `drain_valve`

Creating the Graphical Interface

Forms are the foundation for creating the interface of a Visual Basic application. You can use forms to add windows and dialog boxes to your Visual Basic application. You can also use them as containers for items that are not a visible part of the application's interface. For example, you might have a form in your application that holds a timer object.

The first step in building a Visual Basic application is to create the forms that are the basis for your application's interface. Then you create the objects that make up the interface on the forms. This section assumes that you have a Visual Basic project (see “Creating a New Microsoft® Visual Basic® Project” on page 4-19). For this first application, you will use four types of controls from the toolbox:

- Button
- Timer
- Label
- Scroll bar

- 1 Open `xpc_tank1_COM.vbp`.
- 2 On the left, from the **General** tool panel, click and drag the **Button** icon  to the form to create a button.
- 3 Repeat for a second button.
- 4 If you want to view signal data on the host, return to the **General** tool panel and click and drag the **Timer** icon  to the form to create a timer.
- 5 If you want to view signal data on the host, add a **Label** control to the form. Return to the **General** tool panel and click and drag the **Label** icon  to the form to create a label.
- 6 If you want to be able to vary the parameter input to the target, return to the **General** tool panel and click and drag the **HScrollBar** icon  to the form.
- 7 Next, name your new form objects. Right-click the first button and select **Properties**. This brings up the Properties dialog box. In the **Caption** box, enter `Load`. Repeat for the second button, but enter `Start`. Repeat for the third button, but enter `Stop`. (If you are unsure about how to work with properties, refer to the procedure “Setting Properties” on page 4-25.)

After you name your new form objects and set whatever other parameters you want (for example, if you use a timer you must increase the `Interval`

parameter), you can write the code behind these objects using the Visual Basic code editor window (refer to “Writing Code” on page 4-26).

Note If you add a timer, remember to increase the interval of the timer to a value greater than the default value of 0. Right-click the timer and select **Properties**. This brings up the Properties dialog box. In the **Interval** box, enter a value greater than 0, for example, 100.

Setting Properties

This procedure describes how to set properties for the Visual Basic objects you created on your form. If you already know how to set properties for Visual Basic objects, proceed to “Writing Code” on page 4-26.

The **Properties** window in the following figure provides an easy way to set properties for the objects on a form. To open the **Properties** window, choose the **Properties Window** command from the **View** menu, click the **Properties Window** button on the toolbar, or use the context menu for the control.

The **Properties** window consists of the following elements:

- Object box — Displays the name of the object for which you can set properties. Click the arrow to the right of the object box to display the list of objects for the current form.
- Sort tabs — Choose an alphabetic listing of properties or a hierarchical view divided by logical categories, such as those dealing with appearance, fonts, or position.
- Properties list — The left column displays the properties for the selected object. You can edit and view settings in the right column.

To set properties from the **Properties** window,

- 1 From the **View** menu, choose **Properties**, or click the **Properties** button on the toolbar.

The **Properties** window displays the settings for the selected form or control.

- 2 From the properties list, select the name of a property.
- 3 In the right column, type or select the new property setting.

Enumerated properties have a predefined list of settings. You can display the list by clicking the down arrow at the right of the settings box, or you can cycle through the list by double-clicking a list item.

You can also set object properties directly in the code by using the following dot notation: `Object.propertyname=value`.

Writing Code

The code editor window is where you write Visual Basic code for your application. Code consists of language statements, constants, and declarations. Using the code editor window, you can quickly view and edit the code in your application.

The code editor window has three panes. The top leftmost pane is the object list box. It is a drop-down list that contains the form controls in your project, plus a general section for generic declarations. The top rightmost pane contains a procedure list box. For the selected or active control in the object list box, the procedure list box displays the available procedures, or events. Visual Basic predefines the possible procedures. The third pane contains the code for the Visual Basic application.

In the general declarations section, declare a reference to the xPC Target COM objects that you are using to interface with the xPC Target objects. The following are the objects you need to declare:

- `xPCProtocol` — Reference the classes corresponding to the target computer running the target application and initialize the xPC Target API dynamic link library. At a minimum, you must declare this object.
- `xPCTarget` — Reference the classes for interfacing with the target application. At a minimum, you must declare this object.
- `xPCScope` — If the API application requires signal data, reference the class for interfacing with xPC Target scopes. You need to declare a scope if you want to acquire data from scopes or display data on scopes.

- `model_namept` — This is the COM object for tunable model/application parameters.
- `model_namebio` — This is the COM object for model/target application signals.

Creating the General Declarations

This procedure describes how to create the general object declarations for the `xpctank` (or `xpc_tank1`) model:

- 1 Double-click the form or, from the **View** menu, select **Code**.

The code editor window box opens for the control.

- 2 Select the General object.

- 3 Select **Declarations** in the procedure list box.

A *template* for the declarations procedure is now displayed in the code editor window.

- 4 Enter declarations for the xPC Target COM objects you are using.

```
Public protocol_obj As xPCProtocol
Public target_obj As xPCTarget
Public scope_obj As xPCScopes
```

- 5 Enter declarations for the model-specific COM objects you are using.

```
Public parameters_obj As xpc_tank1pt
Public signals_obj As xpc_tank1bio
```

Creating the Load Procedure

This procedure describes how to program a load target application procedure for the form. You might or might not want to allow users to download target applications to the target computer. However, if you do want to allow this action, you need to provide a control on the GUI for the user to do so. “Creating Event Procedures to Load Applications” on page 4-29 describes how to provide such a control.

- 1 In the project window, double-click the Form object.

The code editor window opens.

- 2 In the procedure list box, select **Load**.
- 3 Create and initialize the objects for the Load method in the form. Note that the following code also checks that the initialization of the `protocol_obj` succeeds. If it does not succeed, an error message is returned and the application will exit.

```
Private Sub Form_Load()  
    Set protocol_obj = New xPCProtocol  
    Set target_obj = New xPCTarget  
    Set scope_obj = New xPCScopes  
    Set parameters_obj = New xpc_tank1pt  
    Set signals_obj = New xpc_tank1bio  
    stat = protocol_obj.Init  
    If stat < 0 Then  
        MsgBox("Could not load api") 'We can no longer continue.  
    End  
    End If  
    stat = protocol_obj.RS232Connect(0, 0)  
    stat = target_obj.Init(protocol_obj)  
    stat = scope_obj.Init(protocol_obj)  
    stat = parameters_obj.Init(protocol_obj.Ref)  
    stat = signals_obj.Init(protocol_obj.Ref)  
End Sub
```

You can add more code to the Load method. This is the minimum code you should enter for this method.

Creating Event Procedures

Code in a Visual Basic application is divided into smaller blocks called *procedures*. Event procedures, such as those you create here, contain code that mainly calls the xPC Target API component methods. For example, when a user clicks a button, that action starts the xPC Target application.

This code is also responsible for the feedback action (such as enabling a timer control, disabling/enabling controls) when an event occurs. An event procedure for a control combines the control's name (specified in the Name property), an underscore (_), and the event name. For example, if you want

a command button named **Command1** to invoke an event procedure when it is clicked, call the procedure `Command1_Click`. The following procedures illustrate how to create event procedures, using the `xpctank` (or `xpc_tank1`) model as an example.

Creating Event Procedures to Load Applications

This procedure describes how to program the command button **Command1** to load an application to the target computer through a serial connection. Provide a procedure like this to allow users to download target applications to the target computer.

- 1 Double-click the form or, from the **View** menu, select **Code**.
- 2 From the object list box, select the name of an object in the active form. (The *active* form is the form that currently has the focus.) For this example, choose the command button **Command1**.
- 3 In the procedure list box, select the name of an event for the selected object. The `Click` procedure is the default procedure for a command button.
- 4 To load the target application, enter the path to the target application. If the target application is in the same folder as the API application, enter ".". Enter the name of the target application without the extension.

```
stat = target_obj.LoadApp(".", "xpc_tank1")
```

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub Command1_Click()  
    stat = target_obj.LoadApp(".", "xpc_tank1")  
End Sub
```

Creating Event Procedures to Start and Stop Applications

This procedure describes how to program the command buttons **Command2** and **Command3** to start and stop an application on a target computer:

- 1 If you are not already in the code editor window, double-click the form or, from the **View** menu, select **Code**.

- 2 From the object list box, select the name of an object in the active form. (The *active* form is the form that currently has the focus.) For this example, choose the command button **Command2**.
- 3 In the procedure list box, select the name of an event for the selected object. Here, select the **Click** procedure.
- 4 To start the target application, select the **StartApp** method for the command button **Command2** (this is the button you named **Start**).

```
stat = target_obj.StartApp
```

- 5 To stop the target application, select the **StopApp** method for the command button **Command3** (this is the button you named **Stop**). Be sure to select the **Click** procedure in the procedure list box.

```
stat = target_obj.StopApp
```

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub Command2_Click()  
    stat = target_obj.StartApp  
End Sub
```

```
Private Sub Command3_Click()  
    stat = target_obj.StopApp  
End Sub
```

Creating Event Procedures to Vary Input Values

You can provide controls to allow users to vary the parameters of their applications. The **Scroll** procedure is one way of varying input. The following code uses the Visual Basic **HScrollBar** object to vary the `water_level` parameter. It takes the value from the **HScrollBar** object and sends that value to the target as a parameter change.

Note This section assumes that you have tagged block parameters and created your own model-specific COM library. Refer to “Getting Parameter IDs with the GetParamIdx Method” on page 4-34 for a description of how to manually perform the equivalent of using tagged parameters.

- 1 If you are not already in the code editor window, double-click the form or, from the **View** menu, select **Code**.
- 2 From the object list box, select the name of an object in the active form. (The *active* form is the form that currently has the focus.) For this example, select the `HScroll11` object.

The cursor jumps to the `HScroll11` object template of the code editor window.

- 3 In the procedure list box, select the name of an event for the selected object. Here, select the `Scroll` procedure.
- 4 Declare the `slideVal` variable as a double. The `slideVal` variable will contain the value of the scroll bar.

```
Dim slideVal(0) As Double
```

- 5 Assign to the `slideVal` variable the result of `Cdbl`. The `Cdbl` function reads the value of an object property. In this example, the object `HScroll11` has the property `slideVal(0)`. `Cdbl` reads the value of `HScroll11.Value` and returns that value to `slideVal`.

```
slideVal(0) = Cdbl(HScroll11.Value)
```

- 6 Set the value of `water_level` to the scroll bar value `slideVal`, which is from `HScrollBar`. The COM object `target_obj` has the method `SetParam`, which has the syntax `SetParam(parIdx, newparVal)`. The `SetParam` method references `parIdx` from the model-specific COM object (type `xpc_tank1pt`). To set the value of `water_level` to the scroll bar value `slideVal`, select `SetParam` and continue typing. A list of the parameters you tagged in the Simulink model then pops up, and you can select the parameter `water_level` and continue typing.

The call to `SetParam` should look like the following:

```
stat = target_obj.SetParam(parameters_obj.water_level,  
slideVal)
```

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub HScroll11_Scroll()  
    Dim slideVal(0) As Double  
  
    slideVal(0) = CDb1(HScroll11.Value)  
    stat = target_obj.SetParam(parameters_obj.water_level,  
slideVal)  
End Sub
```

Creating Event Procedures to Display Signal Values at the Host

You can provide controls to view signal values at the host. To do this, use a combination of the timer and label controls. The following code uses the Visual Basic timer control to display the `water_level` signal on the label control.

Note This section assumes that you have tagged signals and created your own model-specific COM library. Refer to “Getting Signal IDs with the GetSignalIdx Method” on page 4-36 for a description of how to manually perform the equivalent of using tagged signals.

Before you start, check that the `Timer1 Interval` property is greater than 0.

- 1** From the object list box, select the `Timer1` object.
- 2** Assign to the `Label1.Caption` object the value of the `water_level` signal. The COM object `target_obj` has the method `GetSignal(sigNum)`. Reference the `sigNum` parameter by passing it `signals_obj.water_level`. The `CStr` function converts the returned value to a string so that it can be displayed on the `Label1` object.

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub Timer1_Timer()  
    Label1.Caption =  
    CStr(target_obj.GetSignal(signals_obj.water_level))  
End Sub
```

Note Although you add both a timer and label object to the Visual Basic application, only the label appears on the GUI itself when the Visual Basic application is run. The timer is not visible.

Creating Unload and Termination Procedures

You should write Form Unload and Termination procedures to stop and unload the application and to close the communication channel between the host computer and target computer.

Note The Form Unload and Termination procedures must close the communication channel between the host computer and target computer between each run of the GUI application.

The Terminate procedure controls the behavior of the Visual Basic **Run** menu **End** option. The Unload procedure controls the behavior of the Visual Basic **Close** button.

- 1 From the object list box, select the Form object.
- 2 From the procedure list box, select Terminate.
- 3 You are going to close the connection with the target computer, so type `protocol_obj` and select the **Close** method for that object.

```
protocol_obj.Close
```

- 4 From the procedure list box, select Unload.

5 Repeat step

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub Form_Terminate()  
    protocol_obj.Close  
End Sub  
Private Sub Form_Unload(Cancel As Integer)  
    protocol_obj.Close  
End Sub
```

Referencing Parameters and Signals Without Using Tags

The sample code in “Creating Event Procedures to Vary Input Values” on page 4-30 and “Creating Event Procedures to Display Signal Values at the Host” on page 4-32 illustrates how to reference parameters that you tagged before building the Simulink model. This section describes how to reference these same parameters and signals from the COM API application code if you did not opt to tag signals and parameters.

Getting Parameter IDs with the GetParamIdx Method

When working with parameters in the context of varying input values, you use the SetParam and GetParamIdx methods. The SetParam method has the syntax

```
SetParam(ByVal parIdx As Integer, ByRef newparVal As  
System.Array) As Long
```

where **parIdx** is the identifier that corresponds to the parameter you want to set. To obtain the parameter ID, **parIdx**, for SetParam, you need to call the GetParamIdx method. This method has the syntax

```
GetParamIdx(ByVal blockName As String, ByVal paramName As  
String) As Long
```

The following procedure describes how to obtain the GetParamIdx block name and parameter name for the Visual Basic HScrollBar object. You need to reference the block name and parameter from the model_namept.m file.

- 1** Open a DOS window.
- 2** Change the folder to the folder that contains your prebuilt model.
- 3** Open the file `model_namept.m`. For example, you can use the notepad text editor.

```
notepad xpc_tank1pt.m
```

The editor opens for that file. If you are not in the folder in which the `xpc_tank1pt.m` file resides, be sure to type the full path for `xpc_tank1pt.m`.

- 4** Search for and copy the string for the block of the parameter you want to reference. For the `xpc_tank1` example, search for the `SetPoint` block if you want to reference the water level. For example,

```
SetPoint
```

- 5** Return to the code editor window for your project.
- 6** In the line that contains the call to `GetParamIdx`, enter the path for the `blockName` variable.
- 7** Return to the editor window for `model_namept.m`.
- 8** Search for and copy the string for the name of the parameter you are interested in. For example,

```
Value
```

If you do not know the name of the block parameter you are interested in, refer to “Common Block Properties” or “Block-Specific Parameters”.

- 9** Return to the code editor window for your project.
- 10** In the line that contains the call to `GetParamIdx`, enter the path for the `paramName` variable. For example,

```
stat = target_obj.SetParam(target_obj.GetParamIdx  
("SetPoint", "Value"), slideVal)
```

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub HScroll11_Scroll()  
    Dim slideVal(0) As Double  
  
    slideVal(0) = Cdbl(HScroll11.Value)  
    stat =  
target_obj.SetParam(target_obj.GetParamIdx  
("SetPoint", "Value"), slideVal)  
  
End Sub
```

Note, if you want to retrieve the full block path and parameter name of a block, use the `GetParamName` method. The `GetParamName` method returns a variant data type object with two elements. The first element contains the full block path, the second element contains the parameter name. The following example illustrates how to use the `GetParamName` method to get the block path and parameter name:

```
Dim Pname As Variant  
Pname=xpc_tank1.GetParamName(GetParamIdx(Idx)  
BlockPathString=CStr(Pname(0))  
ParameterNameString=CStr(Pname(1))
```

In this example,

- `Idx` is the index to a parameter.
- `BlockPathString` contains the full block path string.
- `ParameterNameString` contains the parameter name string.

Getting Signal IDs with the `GetSignalIdx` Method

When working with signals in the context of displaying signal values, you use the `GetSignal` and `GetSignalIdx` methods. The `GetSignal` method has the syntax

```
GetSignal(sigNum As Long) As Double
```

where `sigNum` is the identifier that corresponds to the signal you want to set.

To obtain the signal ID `sigNum` for `GetSignal`, you call the `GetSignalIdx` method. This method has the syntax

```
GetSignalIdx(sigName As String) As Long
```

The following procedure describes how to obtain the `GetSignalIdx` block name for the Visual Basic timer object. You need to reference the block name and signal from the `model_namebio.m` file.

- 1 Open a DOS window.
- 2 Change the folder to the folder that contains your prebuilt model.
- 3 Open the file `model_namebio.m`. For example,

```
notepad xpc_tank1bio.m
```

The editor opens for that file. If you are not in the folder in which the `xpc_tank1bio.m` file resides, be sure to type the full path for `xpc_tank1bio.m`.

- 4 Search for and copy the string for the block of the signal you want to reference. For the `xpc_tank1` example, search for the `TankLevel` block to reference the tank level. For example,


```
TankLevel
```

- 5 Return to the code editor window for your project.
- 6 In the line that contains the call to `GetSignalIdx`, enter the path for the `SigName` variable.

When you are done, the contents of your code editor window should look similar to the code below:

```
Private Sub Timer1_Timer()  
    Label1.Caption =  
    CStr(target_obj.GetSignal(target_obj.GetSignalIdx("TankLevel"  
)))  
End Sub
```


Testing the Visual Basic Application

While creating your Visual Basic application, you might want to see how the application is progressing. Visual Basic allows you to run your application while still in the Visual Basic project. From the Visual Basic task bar, you can click the **Run** button . Alternatively, you can follow the procedure:

- 1 If you have the MATLAB interface and a target object connected, close the port. For example, at the MATLAB command line, type

```
tg.close
```

- 2 Close any previous version of the application. Only one version of the application should run at a given time.
- 3 From within the project, go to the **Run** menu.
- 4 Select **Start** or **Start with Full Compile**. The **Start** option starts your application immediately. The **Start with Full Compile** option starts the application after compilation.

The form you are working on pops up. Test your application. To stop the application from within Visual Basic, you can click the **End** button  from the task bar. Alternatively, you can go to the **Run** menu and select **End**.

Note

- If the Visual Basic application opens a communication channel between the host and target computers for the target application, close that open channel between test runs of the Visual Basic application. Not doing so can cause subsequent runs of the Visual Basic application to fail.
 - See “Creating Unload and Termination Procedures” on page 4-33 for how to write a procedure to disconnect from the target computer. If you want to return control to the MATLAB interface, close the Visual Basic project first.
-

Building the Visual Basic Application

After you finish designing, programming, and testing your Visual Basic GUI application, build your application. You can later distribute the GUI application to users, who can then use it to work with target applications.

- 1 From within the project, go to the **File** menu.
- 2 Select **Make** `project_name_COM.exe`, where `project_name` is the name of the Visual Basic project you have been working on.
- 3 At the pop-up box, select the folder in which you want to save the executable. Optionally, you can also rename the executable.

The compiler generates the `project_name_COM.exe` file in the specified folder.

Deploying the API Application

This section assumes that you have built your xPC Target application and your Visual Basic xPC Target COM GUI application. If you have not yet done so, refer to “Creating the Target Application and Model-Specific COM Library” on page 4-14 and “Building the Visual Basic Application” on page 4-39, respectively.

When distributing the Visual Basic model application, provide the following files:

- `project_name_COM.exe`, the executable for the Visual Basic application
- `model_name.dlm`

Tip

- Provide `model_name.dlm` if you expect the end user to download the target application to the target computer. Also enable an application load event on the Visual Basic interface (refer to “Creating the Load Procedure” on page 4-27).
 - If you expect the target application to already be loaded on the target computer when the end user runs the Visual Basic GUI application, you might not want him or her to be able to load the target application to the target computer.
-

- `model_nameCOMiface.dll`, if you tag the signals and parameters in the model
- `xpcapiCOM.dll`, the xPC Target COM API dynamic link library
- `xpcapi.dll`, the xPC Target API dynamic link library

All the files must be located in the same folder before the user executes the Visual Basic application.

The end user must know how to register the application-dependent dynamic link libraries (refer to “Registering Dependent Dynamic Link Libraries” on page 4-40).

To run the application and download an xPC Target application, users need to have `project_name_COM.exe` and `model_name.dlm` (if provided) in the same folder.

Registering Dependent Dynamic Link Libraries

This procedure uses `xpc_tank1` as an example.

- 1 Open a DOS window.
- 2 Change the folder to the folder containing the API application files.
- 3 From the folder in which `xpcapiCOM.dll` resides, register the xPC Target COM API DLL by typing

```
regsvr32 xpcapiCOM.dll
```

DOS displays the message

```
DllRegisterServer in xpcapiCOM.dll succeeded
```

Creating a New Visual Basic Project Using Microsoft Visual Studio 8.0

The procedures for the preceding topics apply to Microsoft Visual Studio 6.0 (“Creating a New Microsoft® Visual Basic® Project” on page 4-19). The procedures to use Microsoft Visual Studio 8.0 is similar, with the following exceptions.

- You can open a Microsoft Visual Studio 6.0 project under Microsoft Visual Studio .NET 2003. Microsoft Visual Studio .NET 2003 automatically converts the project.
- If you first create a new Visual Basic project, select **Windows Application** as the template.
- When referencing the xPC Target COM API and model-specific COM libraries, do the following

- 1** From the **Project** menu, click **Add Reference**.

The Add Reference dialog box opens.

- 2** Select the **COM** tab.

- 3** Scroll down the **Component Name** list to the bottom and select the **xPC Target API COM Type Library** item.

- 4** Click **Select**.

xPC Target API COM Type Library appears in the **Selected Components** pane.

- 5** Click **OK**.

- When creating a reference to the xPC Target interface objects, include the COM library. The following illustrates example code on how to reference these objects in Microsoft Visual Studio .NET 2003 and Microsoft Visual Studio 6.0:

Microsoft Visual Studio .NET 2003

```
Public protocol_obj As XPCAPICOMLib.xPCProtocol
Public target_obj As XPCAPICOMLib.xPCTarget
Public scope_obj As XPCAPICOMLib.xPCScopes
```

Microsoft Visual Studio 6.0

```
Public protocol_obj As xPCProtocol
Public target_obj As xPCTarget
Public scope_obj As xPCScopes
```

- When creating an instance of the xPC Target interface objects, include the COM library. The following illustrates example code on how to create an instance of these objects in Microsoft Visual Studio .NET 2003 and Microsoft Visual Studio 6.0:

Microsoft Visual Studio .NET 2003

```
protocol_obj = New XPCAPICOMLib.xPCProtocol
target_obj = New XPCAPICOMLib.xPCTarget
scope_obj = New XPCAPICOMLib.xPCScopes
```

Microsoft Visual Studio 6.0:

```
Set protocol_obj = New xPCProtocol
Set target_obj = New xPCTarget
Set scope_obj = New xPCScopes
```

- Microsoft Visual Studio .NET 2003 builds applications into the **bin** folder of your project area. You cannot choose another location to place your executable.
- When distributing the Visual Basic model application to users, provide the following files in addition to those listed in “Deploying the API Application” on page 4-39:
 - Interop.model_nameACOMIFACELib.dll
 - Interop.XPCAPICOMLib.dll

xPC Target API Examples

- “Visual Basic GUI Using .NET” on page 5-2
- “Visual Basic GUI Using COM” on page 5-5
- “Command Line Scripts Using COM API” on page 5-8

Visual Basic GUI Using .NET

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

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

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

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

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

In this section...
“Before Starting” on page 5-2
“Accessing the Demo Project Solution” on page 5-3
“Rebuilding the Demo Project Solution” on page 5-3
“Using the Demo Executable” on page 5-4

Before Starting

To use the Demo solution, you need

- A target computer running a current xPC Target kernel
- A host computer running the MATLAB software interface, connected to the target computer via RS-232 or TCP/IP
- A target application loaded on the target computer

The xPC Target product ships with an executable version of the example. If you want to rebuild the Demo solution, or if you want to write your own custom GUIs like this one, you need Microsoft Visual Basic .NET installed on the host computer.

Note The xPC Target software allows you to create applications, such as GUIs, to interact with a target computer with COM API functions. “xPC Target COM API” on page 1-7 describes this in detail. To deploy a GUI application to other host computer systems that do not have your licensed copy of the xPC Target product, you need the xPC Target Embedded Option. If you do not have the xPC Target Embedded Option and would like to deploy your GUI application, contact your MathWorks representative.

Accessing the Demo Project Solution

To access the Demo solution,

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

The Microsoft Development Environment for Visual Basic application starts.

- 4 In the **Solution Explorer** pane, double-click `Form1.vb` to display the Demo solution form.

The form is displayed. You can inspect the layout of the example.

- 5 To inspect the form code, select the **View** menu Code option.

The Visual Basic code for the form is displayed.

Rebuilding the Demo Project Solution

To rebuild the Demo solution,

- 1 Double-click `demo.sln`.

The Microsoft Development Environment for Visual Basic application starts.

- 2 Select the **Build** menu Build Solution option.

Using the Demo Executable

To use the Demo solution executable,

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

The GUI is displayed.

Visual Basic GUI Using COM

The Microsoft Visual Basic 6.0 `sf_car_xpc` example illustrates how to create a custom GUI that connects to a target computer. The files for this example are located in

```
C:\matlabroot\toolbox\rtw\targets\xpc\api\VisualBasic\Models\ -  
sf_car_xpc*
```

This application interfaces with the xPC Target application `sf_car_xpc.dlm`, built from the Simulink model `sf_car_xpc`. This model simulates an automatic transmission control system composed of modules that represent the engine, transmission, and vehicle, with an additional logic block to control the transmission ratio. User inputs to the model are in the form of throttle (%) and brake torque (pound-foot).

This example illustrates how you can use the COM API to create a GUI that

- Connects to the target computer via an RS-232 or TCP/IP connection
- Loads the `sf_car_xpc.dlm` target application to the target computer
- Starts and starts the target application engine
- Edits the stop time of the target application
- Edits the sample time of the target application
- Displays the speed, RPM, and gear of the target application engine

Note For detailed information on the project, see the `readme.txt` files located in `C:\matlabroot\toolbox\rtw\targets\xpc\api\VisualBasic\Models\sf_car_xpc*`.

In this section...
“Before Starting” on page 5-6
“Accessing the <code>sf_car_xpc</code> Project” on page 5-6

In this section...

“Rebuilding the sf_car_xpc Project” on page 5-7

“Using the sf_car_xpc Executable” on page 5-7

Before Starting

To use the sf_car_xpc project, you need

- A target computer running a current xPC Target kernel
- A host computer running the MATLAB interface, connected to the target computer via RS-232 or TCP/IP

The xPC Target product ships with an executable version of the sf_car_xpc project. If you want to rebuild the sf_car_xpc project, you need Microsoft Visual Basic 6.0 Professional installed on the host computer. If you want to view or edit the model, you need to have the Stateflow® product installed on the host computer.

Note The xPC Target environment allows you to create applications, such as GUIs, to interact with a target computer with COM API functions. “xPC Target COM API” on page 1-7 describes this in detail. To deploy a GUI application to other host computer systems that do not have your licensed copy of the xPC Target product, you need the xPC Target Embedded Option license. If you do not have the xPC Target Embedded Option license and would like to deploy your GUI application, contact your MathWorks representative.

Accessing the sf_car_xpc Project

To access the sf_car_xpc project,

- 1** Copy the contents of the VisualBasic folder to a writable folder of your choice.
- 2** Change folder to the one that contains your copy of the sf_car_xpc project.
- 3** Double-click the Visual Basic project. For example, double-click sf_car_xpc_COM.vbp.

The Microsoft Visual Basic application starts.

4 In the right **Project** pane, expand the Forms folder.

5 Double-click the form you want to look at.

The form is displayed. You can inspect the layout of it.

6 To inspect the form code, select the **View** menu **Code** option.

The Visual Basic code for the form is displayed.

Rebuilding the sf_car_xpc Project

To rebuild the sf_car_xpc project,

1 Double-click the Visual Basic project. For example, double-click sf_car_xpc_COM.vbp.

The Microsoft Visual Basic application starts.

2 Select the **File** menu **Make sf_car_xpc.exe**.

Using the sf_car_xpc Executable

To use the sf_car_xpc project executable,

1 Change folder to the one that contains your copy of the sf_car_xpc project.

2 Change folder to the bin folder.

3 Double-click sf_car_xpc.exe.

The GUI is displayed.

Command Line Scripts Using COM API

In this section...

“Tcl/Tk Scripts” on page 5-8

“Required Tcl/Tk Software” on page 5-9

“Using the Scripts” on page 5-9

Tcl/Tk Scripts

The Tcl/Tk examples are scripts that illustrate how to directly access xPC Target COM API functions through a command-line interpreter like Tcl/Tk. With Tcl/Tk, you can:

- Write simple command-line scripts that communicate with a target computer and the target application downloaded on that target computer.
- Write simple GUIs that you can use to interact with a target application downloaded on a target computer.

The files for this scripts are located in

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

- `xpcapi.dll` — The xPC Target API DLL file. This file must be in the current (pwd) folder. Alternatively, you can copy the file `xpcapi.dll` into the Windows system folder.
- `xpcbase.tcl` — Contains utility procedures used by the other scripts in the series
- `xpclists.tcl` — Generates a list of signals or parameters for the target application currently loaded on the target computer
- `xpcload.tcl` — Loads the specified target application to the connected target computer
- `xpcoutputlog.tcl` — Reads log data from the target computer and plots the data on the host computer
- `xpcstart.tcl` — Starts the target application loaded on the target computer

- `xpcstop.tcl` — Stops the target application loaded on the target computer
- `xpctargetping.tcl` — Tests the communication between the host and target computers
- `xpctargetscope.tcl` — Creates a simple GUI that enables you to add and control a target scope
- `xpctune.tcl` — Creates a simple GUI slider that enables you to manipulate a parameter value for the target computer application

Required Tcl/Tk Software

To use these Tcl/Tk scripts, or to write your own Tcl/Tk scripts, you need

- An installation of a Tcl/Tk distribution on the host computer.
- An add-on package to the Tcl/Tk interpreter so that the scripts can access the COM API objects. For example, the `tcom` package was used to create the example scripts in the `C:\matlabroot\toolbox\rtw\targets\xpc\api\tcltk` folder.
- The `math::statistics` package. This package is required for the `xpcoutputlog.tcl` file.

Note There are Tcl/Tk distributions that include required and useful packages for use with the xPC Target software. For example, the Tcl/Tk distribution at <http://www.activestate.com/activetcl> contains these packages.

Using the Scripts

The top of each Tcl/Tk script file contains directions on how to use each Tcl/Tk script. In general:

- 1** Copy the contents of the `tcltk` folder to a writable folder of your choice.
- 2** Change folder to the one that contains your copy of the Tcl/Tk script files.
- 3** Start your Tcl/Tk interpreter.
- 4** Load the Tcl/Tk script with the `source` command. For example,

```
source xpctargetping.tcl
```

5 Run the loaded script. For example,

```
xpctargetping 192.168.0.10 22222
```

The selected script executes. In this example, `xpctargetping.tcl` tests the communication between the host and target computer and returns a success or failure message.

xPC Target API Reference for Microsoft .NET Framework

- “xPC Target API for Microsoft .NET Framework Classes” on page 6-2
- “xPC Target API for Microsoft .NET Framework — Alphabetical List” on page 6-7

xPC Target API for Microsoft .NET Framework Classes

Namespace: `MathWorks.xPCTarget.FrameWork`

In this section...
“Target Computers” on page 6-2
“Target Applications” on page 6-3
“Scopes” on page 6-3
“Parameters” on page 6-4
“Signals” on page 6-5
“Data Logs” on page 6-5
“File Systems” on page 6-6
“Errors” on page 6-6

Target Computers

<code>ConnectCompletedEventArgs</code> Class	Data for the event of connecting to the target computer
<code>DisconnectCompletedEventArgs</code> Class	Data for the event of disconnecting from target computer
<code>GetDataCompletedEventArgs</code> Class	Data for the event of completing a data access
<code>LoadCompletedEventArgs</code> Class	Data for the event of loading a target application on the target computer
<code>RebootCompletedEventArgs</code> Class	Data for the event of rebooting the target computer
<code>UnloadCompletedEventArgs</code> Class	Data for the event of unloading the target application from the target computer
<code>xPCProtocol</code> Enumerated Data Type	Host computer and target computer communication medium

xPCRS232BaudRate Enumerated Data Type	Serial communication baud rate
xPCRS232Comport Enumerated Data Type	Serial communication port
xPCTargetPC Class	Access xPCTargetPC class

Target Applications

xPCApplication Class	Access to target application loaded on target computer
xPCAppStatus Enumerated Data Type	Target application status return values

Scopes

SCDISPLAYMODE Enumerated Data Type	Target scope display mode values
SCFILEMODE Enumerated Data Type	Write mode values for when file allocation table entry is updated
SCSTATUS Enumerated Data Type	Scope status values
SCTRIGGERMODE Enumerated Data Type	Scope trigger mode values
SCTRIGGERSLOPE Enumerated Data Type	Scope trigger slope values
SCTYPE Enumerated Data Type	Scope type
xPCFileScope Class	Access to file scopes
xPCFileScopeCollection Class	Collection of xPCFileScope objects
xPCFileScopeSignal Class	Access to file scope signals
xPCFileScopeSignalCollection Class	Collection of xPCFileScopeSignal objects
xPCHostScope Class	Access to host scopes

xPCHostScopeCollection Class	Collection of xPCHostScope objects
xPCHostScopeSignal Class	Access to host scope signals
xPCHostScopeSignalCollection Class	Collection of xPCHostScopeSignal objects
xPCScope Class	Access xPCScope class
xPCScopeCollectionEventArgs Class	Data for the event of adding a scope to a scope collection
xPCScopeRemCollectionEventArgs Class	Data for the event of removing a scope from a scope collection
xPCScopes Class	Access scope objects
xPCScopeSignalCollectionEventArgs Class	Data for the event of adding a signal to a scope signal collection
xPCTargetScope Class	Access to target scopes
xPCTargetScopeCollection Class	Collection of xPCTargetScope objects
xPCTargetScopeSignalCollection Class	Collection of xPCHostScopeSignal objects
xPCTgtScopeSignal Class	Access to target scope signals

Parameters

CancelPropertyNotificationEventArgs Class	Data for the event of cancelling a property value change
GetParamCompletedEventArgs Class	Data for the event of completing a parameter access
PropertyNotificationEventArgs Class	Data for the event of changing property values
SetParamCompletedEventArgs Class	Data for the event of setting a parameter value
xPCParameter Class	Single run-time tunable parameter
xPCParameters Class	Access run-time parameters

Signals

xPCSignal Class	Access signal objects
xPCSignals Class	Access signal objects

Data Logs

GetFileScSignalDataObjectCompletedEventArgs Class	Data for the event of completing a data access to a file scope signal object
GetHostScSignalDataObjectCompletedEventArgs Class	Data for the event of completing a data access to a host scope signal object
GetLogDataCompletedEventArgs Class	Data for the event of completing a data access to a data logging object
xPCAppLogger Class	Access to target application loggers
xPCDataFileScSignalObject Class	Object that holds logged file scope signal data
xPCDataHostScSignalObject Class	Object that holds logged host scope signal data
xPCDataLoggingObject Class	Object that holds logged data
xPCLog Class	Base xPCLog class
xPCLogMode Enumerated Data Type	Specify log mode values
xPCLogType Enumerated Data Type	Logging type values
xPCOutputLogger Class	Access to output logger
xPCStateLogger Class	Access to state log
xPCTETLogger Class	Access to TET logger
xPCTimeLogger Class	Access to output log

File Systems

xPCDirectoryInfo Class	Access folders and subfolders of target computer file system
xPCDriveInfo Class	Information for target computer drive
xPCFileInfo Class	Access to file and xPCFileStream objects
xPCFileMode Enumerated Data Type	Open file with permissions
xPCFileStream Class	Access xPCFileStream objects
xPCFileSystem Class	File system drives and folders
xPCFileSystemInfo Class	File system information

Errors

xPCException Class	Information for xPCException
xPCExceptionReason Enumerated Data Type	Exception reasons

xPC Target API for Microsoft .NET Framework – Alphabetical List

Namespace: `MathWorks.xPCTarget.FrameWork`

xPCFileScopeCollection.Add

Purpose Create xPCFileScope object with the next available scope ID as key

Syntax

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

Description Class: xPCFileScopeCollection Class

Method

Syntax Language: C#

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

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

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

Purpose

Add signals to file scope

Syntax

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

Description

Class: xPCFileScopeSignalCollection Class

Method

Syntax Language: C#

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

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

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

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

Exception

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

xPCHostScopeCollection.Add

Purpose Create xPCHostScope object with the next available scope ID as key

Syntax

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

Description Class: xPCHostScopeCollection Class

Method

Syntax Language: C#

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

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

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

Exception

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

Purpose Add signals to host scope

Syntax

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

Description **Class:** xPCHostScopeSignalCollection Class

Method

Syntax Language: C#

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

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

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

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

Exception

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

xPCTargetScopeCollection.Add

Purpose Create xPCTargetScope object

Syntax

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

Description Class: xPCTargetScopeCollection Class

Method

Syntax Language: C#

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

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

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

xPCTargetScopeSignalCollection.Add

Purpose

Create xPCTargetScopeSignal object

Syntax

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

Description

Class: xPCTargetScopeSignalCollection Class

Method

Syntax Language: C#

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

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

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

`public IList<xPCTgtScopeSignal> Add(int[] sigIds)` creates an ILLIST 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.

xPCTargetScopeSignalCollection.Add

Exception

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

Purpose Close current stream

Syntax `public void Close()`

Description **Class:** xPCFileStream Class

Method

Syntax Language: C#

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

Exception

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

xPCTargetPC.Connect

Purpose Establish connection to target computer

Syntax `public void Connect()`

Description Class: xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

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

Purpose Asynchronous request for target computer connection

Syntax `public void ConnectAsync()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.ConnectCompleted

Purpose Event when asynchronous connect operation completes

Syntax `public event ConnectCompleted ConnectCompleted`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

`public event ConnectCompleted ConnectCompleted` occurs when an asynchronous connect operation completes.

Purpose Event after asynchronous connect operation completes

Syntax `public event EventHandler Connected`

Description **Class:** xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Connected` occurs after asynchronous connect operation completes.

xPCTargetPC.Connecting

Purpose Event before asynchronous connect operation completes

Syntax `public event EventHandler Connecting`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Connecting` occurs before asynchronous connect operation completes.

Purpose Copy specified file from target computer file system to new location on host file system

Syntax `public FileInfo CopyToHost(string HostDestFileName)`

Description **Class:** xPCFileInfo Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
ArgumentException	<i>HostDestFileName</i> is empty, contains only white spaces, or contains invalid characters.
ArgumentNull-Exception	<i>HostDestFileName</i> is NULL reference.
NotSupported-Exception	<i>HostDestFileName</i> contains a colon (:) in the middle of the string.
PathTooLong-Exception	The specified path, file name, or both in <i>HostDestFileName</i> exceed the system-defined maximum length. For example, on Windows platforms, path names must be less than 248 characters. File names must be less than 260 characters.
SecurityException	Caller does not have required permission.
UnauthorizedAccess-Exception	System does not allow access to <i>HostDestFileName</i> .
xPCException	When problem occurs, query xPCException object Reason property.

xPCFileInfo.Create

Purpose Create file in specified path name

Syntax `public xPCFileStream Create()`

Description Class: xPCFileInfo Class

Method

Syntax Language: C#

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

Exception

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

Purpose Create folder

Syntax `public xPCDirectoryInfo CreateDirectory(string path)`

Description **Class:** xPCFileSystem Class

Method

Syntax Language: C#

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

Exception

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

xPCDirectoryInfo.Create

Purpose Create folder

Syntax `public void Create()`

Description Class: xPCDirectoryInfo Class

Method

Syntax Language: C#

`public void Create()` creates a folder.

Purpose Delete current file or folder

Syntax `public abstract void Delete()`

Description **Class:** xPCFileSystemInfo Class

Method

Syntax Language: C#

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

xPCDirectoryInfo.Delete

Purpose Delete empty xPCDirectoryInfo object

Syntax `public override void Delete()`

Description Class: xPCDirectoryInfo Class

Method

Syntax Language: C#

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

Purpose Permanently delete file on target computer

Syntax `public override void Delete()`

Description **Class:** xPCFileInfo Class

Method

Syntax Language: C#

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

Exception

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

xPCTargetPC.Disconnect

Purpose Close connection

Syntax `public void Disconnect()`

Description Class: xPCTargetPC Class

Method

Syntax Language: C#

`public void Disconnect()` closes the connection.

Exception

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

Purpose Begin asynchronous request to disconnect from target computer

Syntax `public void DisconnectAsync()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.DisconnectCompleted

Purpose Event when asynchronous disconnect operation completes

Syntax `public event DisconnectCompletedEventHandler DisconnectCompleted`

Description **Class:** xPCTargetPC Class

Event

Syntax Language: C#

`public event DisconnectCompletedEventHandler DisconnectCompleted` occurs when asynchronous disconnect operation completes.

Purpose	Event after asynchronous disconnect operation completes
Syntax	<code>public event EventHandler Disconnected</code>
Description	<p>Class: xPCTargetPC Class</p> <p>Event</p> <p>Syntax Language: <i>C#</i></p> <p><code>public event EventHandler Disconnected</code> occurs after asynchronous disconnect operation completes.</p>

xPCTargetPC.Disconnecting

Purpose Event before asynchronous disconnect operation completes

Syntax `public event EventHandler Disconnecting`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Disconnecting` occurs before asynchronous disconnect operation completes.

Purpose Clean up resources

Syntax `public void Dispose()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

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

xPCTargetPC.Disposed

Purpose Event after disposal of used resources completes

Syntax `public event EventHandler Disposed`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Disposed` occurs after disposal of used resources completes.

xPCFileSystem.GetCurrentDirectory

Purpose Current working folder for target application

Syntax `public string GetCurrentDirectory()`

Description Class: xPCFileSystem Class

Method

Syntax Language: C#

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

Exception

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

xPCDataLoggingObject.GetData

Purpose Logged data from target computer

Syntax `public double[] GetData()`

Description Class: xPCDataLoggingObject Class

Method

Syntax Language: *C#*

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

Purpose Logged file scope signal data from target computer

Syntax `public double[] GetData()`

Description **Class:** xPCDataFileScSignalObject Class

Method

Syntax Language: *C#*

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

xPCDataHostScSignalObject.GetData

Purpose Logged host scope signal data from target computer

Syntax `public double[] GetData()`

Description Class: xPCDataHostScSignalObject Class

Method

Syntax Language: *C#*

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

Purpose Logged data from target computer without blocking calling thread

Syntax

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

Description **Class:** xPCDataLoggingObject Class

Method

Syntax Language: C#

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

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

xPCDataFileScSignalObject.GetDataAsync

Purpose File scope signal logged data from target computer without blocking calling thread

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

Description Class: xPCDataFileScSignalObject Class

Method

Syntax Language: C#

`public void GetDataAsync()` copies the file scope signal logged data from the target computer without blocking the calling thread. This is an asynchronous request.

`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 completes.

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCDataHostScSignalObject.GetDataAsync

Purpose Host scope signal logged data from target computer without blocking calling thread

Syntax

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

Description **Class:** xPCDataHostScSignalObject Class

Method

Syntax Language: C#

`public void GetDataAsync()` copies the host scope signal logged data from the target computer without blocking the calling thread. This is an asynchronous request.

`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 completes.

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCDataLoggingObject.GetDataCompleted

Purpose	Event when copy logged data operation completes
Syntax	<code>public event GetDataCompletedEventHandler GetDataCompleted</code>
Description	<p>Class: xPCDataLoggingObject Class</p> <p>Event</p> <p>Syntax Language: C#</p> <p><code>public event GetDataCompletedEventHandler GetDataCompleted</code> occurs when the asynchronous copy logged data operation completes.</p>

xPCDataFileScSignalObject.GetDataCompleted

Purpose	Event when copy logged file scope signal data operation completes
Syntax	<pre>public event GetFileScSignalDataCompletedEventHandler GetDataCompleted</pre>
Description	<p>Class: xPCDataFileScSignalObject Class</p> <p>Event</p> <p>Syntax Language: C#</p> <pre>public event GetFileScSignalDataCompletedEventHandler</pre> <p><code>GetDataCompleted</code> occurs when the asynchronous copy file scope signal logged data operation completes.</p>

xPCDataHostScSignalObject.GetDataCompleted

Purpose	Event when copy logged host scope signal data operation completes
Syntax	<code>public event GetDataCompletedEventHandler GetDataCompleted</code>
Description	<p>Class: xPCDataHostScSignalObject Class</p> <p>Event</p> <p>Syntax Language: C#</p> <p><code>public event GetDataCompletedEventHandler GetDataCompleted</code> occurs when the asynchronous copy host scope signal logged data operation completes.</p>

Purpose Subfolders of current folder

Syntax `public xPCDirectoryInfo[] GetDirectories()`

Description **Class:** xPCDirectoryInfo Class

Method

Syntax Language: C#

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

xPCFileSystem.GetDrives

Purpose Drive names for the logical drives on the target computer

Syntax `public xPCDriveInfo[] GetDrives()`

Description Class: xPCFileSystem Class

Method

Syntax Language: C#

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

Exception

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

Purpose File list from current folder

Syntax `public xPCFileInfo[] GetFiles()`

Description **Class:** xPCDirectoryInfo Class

Method

Syntax Language: C#

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

xPCDirectoryInfo.GetFileSystemInfos

Purpose File system information for files and subfolders in folder

Syntax `public xPCFileSystemInfo[] GetFileSystemInfos()`

Description Class: xPCDirectoryInfo Class

Method

Syntax Language: *C#*

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

Purpose Number of dimensions

Syntax `public double[] GetParam()`

Description **Class:** xPCParameter Class

Method

Syntax Language: C#

`public double[] GetParam()` gets number of dimensions for the parameter. It returns these dimensions as an array of doubles.

Exception

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

xPCParameter.GetParamAsync

Purpose Asynchronous request for parameter values from target computer

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

Description **Class:** xPCParameter Class

Method

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCParameter.GetParamCompleted

Purpose	Event when get parameter operation completes
Description	<p>Class: xPCParameter Class</p> <p>Event</p> <p>Syntax Language: C#</p> <pre>public event GetParamCompletedEventHandler GetParamCompleted occurs when an asynchronous get parameter operation completes.</pre>

xPCSignals.GetSignals

Purpose List of xPCSignal objects specified by array of signal identifiers

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

Description Class: xPCSignals Class

Method

Syntax Language: C#

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

`public IList<xPCSignal> GetSignals(int[] arrayOfSigId)` returns the list of xPCSignal objects specified by an array of signal identifiers. This method creates an ILLIST 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.

Purpose

Vector of signal values from array

Syntax

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

Description

Class: xPCSignals Class

Method

Syntax Language: C#

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

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

Exception

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

xPCSignal.GetValue

Purpose Value of signal at moment of request

Syntax `public virtual double GetValue()`

Description Class: xPCSignal Class

Method

Syntax Language: C#

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

Exception

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

Purpose Load target application onto target computer

Syntax

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

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

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

Exception

Exception	Condition
ArgumentException	<i>DLMFileName</i> is empty, contains only white spaces, or contains invalid characters.
xPCException	When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.
InvalidOperationException	<i>DLMFileName</i> is a NULL reference (empty in Visual Basic) or an empty string.
NotSupportedException	<i>DLMFileName</i> contains a colon (:) in the middle of the string.
PathTooLongException	The specified path, file name, or both in <i>DLMFileName</i> exceed the system-defined maximum length. For example, on Windows platforms, path names must be less than 248 characters. File names must be less than 260 characters.

xPCTargetPC.Load

Exception	Condition
SecurityException	Caller does not have required permission.
UnauthorizedAccess-Exception	System does not allow access to <i>DLMFileName</i> .

Purpose Begin asynchronous request for loading target application onto target computer

Syntax `public void LoadAsync()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.LoadCompleted

Purpose Event when asynchronous load operation completes

Syntax `public event LoadCompletedEventHandler LoadCompleted`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

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

Purpose Event after target application load completes

Syntax `public event EventHandler Loaded`

Description **Class:** xPCTargetPC Class

Event

Syntax Language: *C#*

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

xPCTargetPC.Loading

Purpose Event before loading of target application completes on target computer

Syntax `public event EventHandler Loading`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

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

xPCParameters.LoadParameterSet

Purpose Load parameter values for target application

Syntax `public void LoadParameterSet(string fileName)`

Description **Class:** xPCParameters Class

Method

Syntax Language: C#

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

Exception

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

CancelPropertyNotificationEventArgs Class

Purpose Data for the event of cancelling a property value change

Syntax

```
public class CancelPropertyNotificationEventArgs : PropertyNotificationEventArgs
```

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

Syntax Language: C#

```
public class CancelPropertyNotificationEventArgs : PropertyNotificationEventArgs provides data for a CancelPropertyNotification event.
```

Properties

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

ConnectCompletedEventArgs Class

Purpose Data for the event of connecting to the target computer

Syntax

```
public class ConnectCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class ConnectCompletedEventArgs : AsyncCompletedEventArgs
```

 provides data for a ConnectCompleted event of xPCTargetPC type.

Properties

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

DisconnectCompletedEventArgs Class

Purpose Data for the event of disconnecting from target computer

Syntax

```
public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class DisconnectCompletedEventArgs : AsyncCompletedEventArgs
```

 provides data for a DisconnectCompleted event of xPCTargetPC type.

Properties

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

GetDataCompletedEventArgs Class

Purpose Data for the event of completing a data access

Syntax

```
public class GetDataCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class GetDataCompletedEventArgs :  
AsyncCompletedEventArgs provides data for GetDataCompleted  
events of various types.
```

Properties

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

GetFileScSignalDataObjectCompletedEventArgs Class

Purpose Data for the event of completing a data access to a file scope signal object

Syntax

```
public class GetFileScSignalDataObjectCompletedEventArgs : GetDataCompletedEventArgs
```

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

Syntax Language: C#

```
public class GetFileScSignalDataObjectCompletedEventArgs : GetDataCompletedEventArgs provides data for GetDataCompleted event of xPCDataFileScSignalObject type.
```

Properties

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

GetFileScSignalDataObjectCompletedEventArgs Class

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

GetHostScSignalDataObjectCompletedEventArgs Class

Purpose Data for the event of completing a data access to a host scope signal object

Syntax

```
public class GetHostScSignalDataObjectCompletedEventArgs : GetDataCompletedEventArgs
```

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

Syntax Language: C#

```
public class GetHostScSignalDataObjectCompletedEventArgs
: GetDataCompletedEventArgs provides data for
ScSignalDataObjectCompleted event of xPCDataHostScSignalObject
type.
```

Properties

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

GetHostScSignalDataObjectCompletedEventArgs Class

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

GetLogDataCompletedEventArgs Class

Purpose Data for the event of completing a data access to a data logging object

Syntax

```
public class GetLogDataCompletedEventArgs : GetDataCompletedEventArgs
```

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

Syntax Language: C#

```
public class GetLogDataCompletedEventArgs :  
GetDataCompletedEventArgs provides data to GetDataCompleted  
event of xPCDataLoggingObject type.
```

Properties

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

GetLogDataCompletedEventArgs Class

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

GetParamCompletedEventArgs Class

Purpose Data for the event of completing a parameter access

Syntax

```
public class GetParamCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class GetParamCompletedEventArgs :  
AsyncCompletedEventArgs provides data for GetParamCompleted  
event of xPCParameter type.
```

Properties

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

LoadCompletedEventArgs Class

Purpose Data for the event of loading a target application on the target computer

Syntax `public class LoadCompletedEventArgs : AsyncCompletedEventArgs`

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

Syntax Language: C#

`public class LoadCompletedEventArgs :`
AsyncCompletedEventArgs provides data for LoadCompleted event of xPCTargetPC type.

Properties

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

PropertyNotificationEventArgs Class

Purpose Data for the event of changing property values

Syntax

```
public class PropertyNotificationEventArgs : PropertyChangedEventArgs
```

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

Syntax Language: C#

```
public class PropertyNotificationEventArgs :  
PropertyChangedEventArgs provides data for PropertyNotification  
event.
```

Properties

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

RebootCompletedEventArgs Class

Purpose Data for the event of rebooting the target computer

Syntax

```
public class RebootCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class RebootCompletedEventArgs : AsyncCompletedEventArgs provides data for RebootCompleted event of xPCTargetPC type.
```

Properties

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

SetParamCompletedEventArgs Class

Purpose Data for the event of setting a parameter value

Syntax

```
public class SetParamCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class SetParamCompletedEventArgs :  
AsyncCompletedEventArgs provides data for SetParamCompleted  
event of xPCParameter type.
```

Properties

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

UnloadCompletedEventArgs Class

Purpose Data for the event of unloading the target application from the target computer

Syntax

```
public class UnloadCompletedEventArgs : AsyncCompletedEventArgs
```

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

Syntax Language: C#

```
public class UnloadCompletedEventArgs : AsyncCompletedEventArgs
```

 provides data for UnloadCompleted event of xPCTargetPC type.

Properties

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

xPCApplication Class

Purpose Access to target application loaded on target computer

Syntax `public sealed class xPCApplication : xPCBaseNotification`

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

Syntax Language: C#

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

Methods

Method	Description
<code>xPCApplication.Start</code>	Start target application simulation
<code>xPCApplication.Stop</code>	Stop target application simulation

Events

Events	Description
<code>xPCApplication.Started</code>	Event after simulation starts from issue of stop command
<code>xPCApplication.Starting</code>	Event before simulation starts from issue of start command
<code>xPCApplication.Stopped</code>	Event after simulation stops from issue of stop command
<code>xPCApplication.Stopping</code>	Event before simulation stops from issue of stop command

Properties

Properties	C# Declaration Syntax	Description	Exception
CPUOverload	<code>public bool CPUOverload {get;}</code>	Get state of CPUOverload.	xPCException — When problem occurs, query xPCException object Reason property.
ExecTime	<code>public double ExecTime {get;}</code>	Get execution time.	xPCException — When problem occurs, query xPCException object Reason property.
Logger	<code>public xPCAppLogger Logger {get;}</code>	Get reference to the application logging object.	
MaximumTeT	<code>public double MaximumTeT {get;}</code>	Get the maximum time. The first element contains the maximum TET number; the second element contains how long it took to achieve the TET time.	xPCException — When problem occurs, query xPCException object Reason property.

xPCApplication Class

Properties	C# Declaration Syntax	Description	Exception
MinimumTet	<code>public double MinimumTet {get;}</code>	Get the minimum time. The first element contains the minimum TET number; the second element contains how long it took to achieve the TET time.	xPCException — When problem occurs, query xPCException object Reason property.
Name	<code>public string Name {get;}</code>	Get the current name of the loaded target application	xPCException — When problem occurs, query xPCException object Reason property.
Parameters	<code>public xPCParameters Parameters {get;}</code>	Get reference to the xPCParameters object.	
SampleTime	<code>public double SampleTime {get; set;}</code>	Get or set Sample time	xPCException — When problem occurs, query xPCException object Reason property.
Scopes	<code>public xPCScopes Scopes {get;}</code>	Get collection of scopes assigned to the application	
Signals	<code>public xPCSignals Signals {get;}</code>	Get reference to xPCSignals object	

xPCApplication Class

Properties	C# Declaration Syntax	Description	Exception
Status	<code>public xPCAppStatus Status {get;}</code>	Get simulation status. See xPCAppStatus Enumerated Data Type.	xPCException — When problem occurs, query xPCException object Reason property.
StopTime	<code>public double StopTime {get; set;}</code>	Get and set stop time	xPCException — When problem occurs, query xPCException object Reason property.
Target	<code>public xPCTargetPC Target {get;}</code>	Get reference to parent xPCTargetPC object.	

xPCAppLogger Class

Purpose Access to target application loggers

Syntax `public class xPCAppLogger : xPCApplicationObject`

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

Syntax Language: C#

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

Properties

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

xPCDataFileScSignalObject Class

Purpose Object that holds logged file scope signal data

Syntax

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

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

Syntax Language: C#

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

 accesses an object that holds logged file scope signal data.

Methods

Method	Description
<code>xPCDataFileScSignalObject.GetData</code>	Get file scope signal data from target computer
<code>xPCDataFileScSignalObject.GetDataAsync</code>	Get file scope logged data from target computer without blocking calling thread

Events

Event	Description
<code>xPCDataFileScSignalObject.EventDataComplete</code>	Event that is raised when file scope signal data operation completes

Properties

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

xPCDataHostScSignalObject Class

Purpose Object that holds logged host scope signal data

Syntax

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

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

Syntax Language: C#

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

Methods

Method	Description
xPCDataHostScSignalObject.LoggedData	Gets logged host scope signal data from target computer
xPCDataHostScSignalObject.LoggedDataAsync	Gets logged data from target computer without blocking calling thread

Events

Event	Description
xPCDataHostScSignalObject.EventDataCompleted	Host scope signal data operation completes

xPCDataHostScSignalObject Class

Properties

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

xPCDataLoggingObject Class

Purpose Object that holds logged data

Syntax

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

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

Syntax Language: C#

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

Methods

Method	Description
xPCDataLoggingObject.GetData	Logged data from target computer
xPCDataLoggingObject.GetDataAsync	Logged data from target computer without blocking calling thread

Events

Event	Description
xPCDataLoggingObject.DataCompleted	Logged data operation completes

Properties

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

xPCDataLoggingObject Class

Property	C# Declaration Syntax	Description
NumSamples	<pre>public int NumSamples {get; set;}</pre>	<p>Get or set number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection. It then has zeroes for the remaining uncollected data. Note what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works with the autorestart setting. If autorestart is enabled, the file scope collects data up to <code>NumSamples</code>, then starts over again, overwriting the buffer. If autorestart is disabled, the file scope collects data only up to <code>NumSamples</code>, then stops.</p>
Startindex	<pre>public int StartIndex {get; set;}</pre>	<p>Get and set the index of the first sample to retrieve from the log.</p>

xPCDirectoryInfo Class

Purpose Access folders and subfolders of target computer file system

Syntax `public class xPCDirectoryInfo : xPCFileSystemInfo`

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

Syntax Language: C#

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

Constructor

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

Methods

Method	Description
<code>xPCDirectoryInfo.Create</code>	Create folder
<code>xPCDirectoryInfo.Delete</code>	Delete empty <code>xPCDirectoryInfo</code> object
<code>xPCDirectoryInfo.GetDirectories</code>	Subfolders of current folder
<code>xPCDirectoryInfo.GetFiles</code>	File list from current folder
<code>xPCDirectoryInfo.GetFilesInfo</code>	File system information for files and subfolders in folder

Properties

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

xPCDriveInfo Class

Purpose Information for target computer drive

Syntax `public class xPCDriveInfo`

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

Syntax Language: `C#`

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

Constructor

Constructor	Description
<code>xPCDriveInfo</code>	Initialize new instance of <code>xPCDriveInfo</code> class

Methods

Method	Description
<code>xPCDriveInfo.Refresh</code>	Synchronize with file drives on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Available-Freespace	<code>public long AvailableFreeSpace {get;}</code>	Indicate amount of available free space on drive.	<code>xPCException</code> — When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.
DriveFormat	<code>public string DriveFormat {get;}</code>	Get name of file system type, such as FAT16 or FAT32.	<code>xPCException</code> — When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.

xPCDriveInfo Class

Property	C# Declaration Syntax	Description	Exception
Name	<code>public string Name {get;}</code>	Get name of drive.	xPCException — When problem occurs, query xPCException object Reason property.
Root-Directory	<code>public xPCDirectoryInfo RootDirectory</code>	Get root folder of drive.	xPCException — When problem occurs, query xPCException object
TotalSize	<code>public long TotalSize {get;}</code>	Get total size of drive in bytes.	xPCException — When problem occurs, query xPCException object Reason property.
VolumeLabel	<code>public string VolumeLabel {get;}</code>	Get volume label of drive.	xPCException — When problem occurs, query xPCException object Reason property.

xPCException Class

Purpose Information for xPCException

Syntax `public class xPCException : Exception, ISerializable`

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

Syntax Language: C#

`public class xPCException : Exception, ISerializable`
accesses information on xPC Target exceptions.

Constructor

Constructor	Description
xPCException	Construct new instance of xPCException class

Properties

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

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

xPCFileInfo Class

Purpose Access to file and xPCFileStream objects

Syntax `public class xPCDriveInfo`

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

Syntax Language: C#

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

Constructor

Constructor	Description
xPCFileInfo	Construct new instance of xPCFileInfo class

Methods

Method	Description
xPCFileInfo.CopyToHost	Event before establishing connection
xPCFileInfo.Create	Create file in specified path name
xPCFileInfo.Delete	Permanently delete file on target computer
xPCFileInfo.Open	Open file
xPCFileInfo.OpenRead	Create read-only xPCFileStream object
xPCFileInfo.Rename	Rename file
xPCFileInfo	Construct new instance of xPCFileInfo class

Properties

Property	C# Declaration Syntax	Description
Directory	<code>public xPCDirectoryInfo Directory {get;}</code>	Get an xPCDirectoryInfo object.

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

xPCFileScope Class

Purpose Access to file scopes

Syntax `public class xPCFileScope : xPCScope`

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

Syntax Language: C#

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

Methods

The xPCFileScope class inherits methods from xPCScope Class.

Events

The xPCFileScope class inherits events from xPCScope Class.

Properties

The xPCFileScope class inherits its other properties from xPCScope Class.

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

xPCFileScope Class

Property	C# Declaration Syntax	Description	Exception
MaxWrite-FileSize	<pre>public uint MaxWriteFileSize {get; set;}</pre>	<p>Get or set the maximum file size in bytes allowed before incrementing to the next file.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified.</p> <p>If the software cannot create additional log files, it overwrites the first log file.</p> <p>This value must be a multiple of <code>WriteSize</code>. Default is 536870912.</p>	<p><code>xPCException</code> — When problem occurs, query <code>xPCException</code> object <code>Reason</code> property.</p>
Signals	<pre>public xPCTarget- ScopeSignalCollection Signals {get;}</pre>	<p>Get collection of file scope signals (<code>xPCFileScopeSignalCollection</code>) assigned to this scope object.</p>	

Property	C# Declaration Syntax	Description	Exception
Trigger-Signal	<pre>public xPCTgtScopeSignal TriggerSignal {get; set;}</pre>	Get or set file scope signal (xPCFileScopeSignal) used to trigger the scope.	xPCException — When problem occurs, query xPCException object Reason property.
WriteSize	<pre>public int WriteSize {get; set;}</pre>	Get or set the unit number of bytes for memory buffer writes. The memory buffer accumulates data in multiples of write size. <i>WriteSize</i> must be multiple of 512.	xPCException — When problem occurs, query xPCException object Reason property.

xPCFileScopeCollection Class

Purpose Collection of xPCFileScope objects

Syntax

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

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

Syntax Language: C#

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

Methods

Method	Description
xPCFileScopeCollection	Create xPCFileScope object with the next available scope ID as key
xPCFileScopeCollection	Refresh
xPCFileScopeCollection	StartAll file scopes in one call
xPCFileScopeCollection	StopAll file scopes in one call

Purpose Access to file scope signals

Syntax `public class xPCFileScopeSignal : xPCScopeSignal`

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

Syntax Language: C#

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

Properties

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

xPCFileScopeSignalCollection Class

Purpose 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
<code>xPCFileScopeSignalCollection.AddSignal</code>	Adds signals to file scope
<code>xPCFileScopeSignalCollection.Refresh</code>	Synchronizes with signals for associated scope on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	<code>public xPCFileScopeSignalItem[string blkpath] {get;}</code>	Get xPCFileScopeSignal object from signal name (<i>blkpath</i>). <i>blkpath</i> is the signal name that represents a signal object added to its parent xPCHostScope object. This property returns the file scope	xPCException — When problem occurs, query xPCException object Reason property.

xPCFileScopeSignalCollection Class

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

xPCFileStream Class

Purpose Access xPCFileStream objects

Syntax `public class xPCFileStream : IDisposable`

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

Syntax Language: C#

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

Constructor

Constructor	Description
xPCFileStream	Construct new instance of xPCFileStream class

Methods

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

Property

Property	C# Declaration Syntax	Description	Exception
Length	<code>public long Length {get;}</code>	Get length of file stream.	xPCException — When problem occurs, query xPCException object Reason property.

Purpose File system drives and folders

Syntax `public class xPCFileSystem`

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

Syntax Language: C#

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

Methods

Method	Description
<code>xPCFileSystem.Create</code>	Create folder
<code>xPCFileSystem.GetCurrentDirectory</code>	Current working folder for target application
<code>xPCFileSystem.GetDrives</code>	Drive names for the logical drives on the target computer
<code>xPCFileSystem.RemoveFile</code>	Remove file name from target computer
<code>xPCFileSystem.SetCurrentDirectory</code>	Current folder

xPCFileSystemInfo Class

Purpose File system information

Syntax `public abstract class xPCFileSystemInfo`

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

Syntax Language: C#

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

Constructor

Constructor	Description
<code>xPCFileSystemInfo</code>	Initialize new instance of <code>xPCFileSystemInfo</code> class

Methods

Method	Description
<code>xPCFileSystemInfo.Delete</code>	Delete current folder

Properties

Property	C# Declaration Syntax	Description
CreationTime	<code>public DateTime CreationTime {get;}</code>	Get creation time of current <code>FileSystemInfo</code> object.
Exists	<code>public abstract bool Exists {get;}</code>	Get value that indicates existence of file or folder.
Extension	<code>public string Extension {get;}</code>	Get string that represents file extension.

xPCFileSystemInfo Class

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

xPCHostScope Class

Purpose Access to host scopes

Syntax `public class xPCHostScope : xPCScope`

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

Syntax Language: C#

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

Methods

The xPCHostScope class inherits methods from xPCScope Class.

Events

The xPCHostScope class inherits events from xPCScope Class.

Properties

The xPCHostScope class inherits its other properties from xPCScope Class.

Property	C# Declaration Syntax	Description	Exception
DateTime-Object	<code>public xPCDataHostScSignalObject DateTimeObject {get;}</code>	Get host scope time data object xPCDataHostScSignalObject associated with this scope.	
Signals	<code>public xPCTargetScopeSignal-</code>	Get collection of host scope signals (xPCHost-	

xPCHostScope Class

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

xPCHostScopeCollection Class

Purpose Collection of xPCHostScope objects

Syntax

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

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

Syntax Language: C#

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

Methods

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

xPCHostScopeSignal Class

Purpose Access to host scope signals

Syntax `public class xPCHostScopeSignal : xPCScopeSignal`

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

Syntax Language: C#

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

Properties

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

xPCHostScopeSignalCollection Class

Purpose Collection of xPCHostScopeSignal objects

Syntax `public class xPCHostScopeSignal : xPCScopeSignal`

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

Syntax Language: C#

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

Methods

Method	Description
<code>xPCHostScopeSignalCollection.CreateAdd</code>	Creates and adds xPCHostScopeSignal object
<code>xPCHostScopeSignalCollection.SyncRefresh</code>	Synchronizes signals for associated host scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	<code>public xPCHostScopeSignal Item[string blkpath] {get;}</code>	Get xPCHostScopeSignal object from signal name (<i>blkpath</i>). <i>blkpath</i> is the signal name that represents a signal object added to its parent xPCHostScope object. This property returns the file scope signal	xPCException — When problem occurs, query xPCException object Reason property.

xPCHostScopeSignalCollection Class

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

xPCLog Class

Purpose Base xPCLog class

Syntax `public abstract class xPCLog : xPCApplicationObject`

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

Syntax Language: C#

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

Properties

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

Purpose Access to output logger

Syntax `public class xPCOutputLogger : xPCLog`

Description **Namespace:** MathWorks.xPCTarget.FrameWork
Syntax Language: C#

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

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

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

xPCParameter Class

Purpose Single run-time tunable parameter

Syntax `public class xPCParameter : xPCApplicationNotificationObject`

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

Syntax Language: C#

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

Methods

Method	Description
<code>xPCParameter.GetParam</code>	Get number of dimensions
<code>xPCParameter.GetParamAsync</code>	Asynchronous request to get parameter values from target computer
<code>xPCParameter.SetParam</code>	Set number of dimensions
<code>xPCParameter.SetParamAsync</code>	Asynchronous request to set parameter values on target computer

Events

Event	Description
<code>xPCParameter.GetParamCompleted</code>	Invoked when a get parameter operation completes
<code>xPCParameter.SetParamCompleted</code>	Invoked when a set parameter operation completes

Properties

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

xPCParameters Class

Purpose Access run-time parameters

Syntax `public class xPCParameters : xPCApplicationObject`

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

Syntax Language: C#

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

Methods

Method	Description
<code>xPCParameters.LoadParameters</code>	Load parameter values for target application
<code>xPCParameters.Refresh</code>	Refresh state of object
<code>xPCParameters.SaveParameters</code>	Save parameter values of target application

Properties

Property	C# Declaration Syntax	Description
<code>NumParameters</code>	<code>public int NumParameters {get;}</code>	Get the total number of tunable parameters in the target application.
<code>Item</code>	<code>public xPCParameter Item[int paramIdx] {get;}</code> or <code>public xPCParameter Item[string blkName, string paramName] {get;}</code>	Return reference to <code>xPCParameter</code> object specified by its parameter identifier (<i>paramIdx</i>) or parameter name (<i>paramname</i>). <i>paramIdx</i> is a 32-bit integer parameter identifier that represents the actual signal.

xPCParameters Class

Property	C# Declaration Syntax	Description
		<p><i>blkName</i> is a string that specifies the block path name for the actual block that contains the parameter.</p> <p><i>paramName</i> is a string that specifies the parameter name.</p> <p>This method returns the xPCParameter object that represents the actual parameter.</p>

xPCScope Class

Purpose Access xPCScope class

Syntax

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

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

Syntax Language: C#

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

 initializes a new instance of the xPCScope class.

Methods

Method	Description
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 scope receives start command
xPCScope.ScopeStarting	Event before scope completes starting
xPCScope.ScopeStopped	Event after scope receives stop command
xPCScope.ScopeStopping	Event before scope completes stopping

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.

xPCScope Class

Property	C# Declaration Syntax	Description	Exception
NumSamples	<pre>public int NumSamples {get; set;} Num</pre>	<p>Get or set number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection. It then has zeroes for the remaining uncollected data. Note what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works with the autorestart setting. If autorestart is enabled, the file scope collects data up to NumSamples, then starts over again, overwriting the buffer. If autorestart is disabled, the file scope collects data only up to NumSamples, then stops.</p>	<p>xPCException — When problem occurs, query xPCException object Reason property.</p>
ScopeId	<pre>public int ScopeId {get;}</pre>	<p>A numeric index, unique for each scope.</p>	

Property	C# Declaration Syntax	Description	Exception
Status	<pre>public SCSTATUS Status {get;}</pre>	<p>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'.</p>	<p>xPCException — When problem occurs, query xPCException object Reason property.</p>
TriggerAnySignal	<pre>public int TriggerAnySignal {get; set;}</pre>	<p>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.</p>	<p>xPCException — When problem occurs, query xPCException object Reason property.</p>
TriggerLevel	<pre>public double TriggerLevel {get; set;}</pre>	<p>Get or set trigger level. If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. You can cross the trigger level with either a rising or falling signal.</p>	<p>xPCException — When problem occurs, query xPCException object Reason property.</p>

xPCScope Class

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

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

xPCScopeCollectionEventArgs Class

Purpose Data for the event of adding a scope to a scope collection

Syntax `public class xPCScopeCollectionEventArgs : EventArgs`

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

Syntax Language: C#

`public class xPCScopeCollectionEventArgs : EventArgs`
provides data for Added event of various types.

Properties

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

xPCScopeRemCollectionEventArgs Class

Purpose Data for the event of removing a scope from a scope collection

Syntax `public class xPCScopeRemCollectionEventArgs : EventArgs`

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

Syntax Language: C#

`public class xPCScopeRemCollectionEventArgs : EventArgs`
provides data for ScopeRemCollection event of various types.

Properties

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

xPCScopeSignalCollectionEventArgs Class

Purpose Data for the event of adding a signal to a scope signal collection

Syntax `public class xPCScopeSignalCollectionEventArgs : EventArgs`

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

Syntax Language: C#

`public class xPCScopeSignalCollectionEventArgs : EventArgs`
provides data for the Add event of xPCScopeSignalCollection type.

Properties

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

Purpose Access scope objects

Syntax `public class xPCScopes : xPCApplicationObject`

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

Syntax Language: C#

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

Methods

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

Properties

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

xPCSignal Class

Purpose Access signal objects

Syntax `public class xPCSignal : xPCApplicationObject`

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

Syntax Language: C#

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

Methods

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

Properties

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

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

xPCSignals Class

Purpose Access signal objects

Syntax `public class xPCSignals : xPCApplicationObject`

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

Syntax Language: C#

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

Methods

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

Properties

Property	C# Declaration Syntax	Description	Exception
<code>NumSignals</code>	<code>public int NumSignals {get;}</code>	Get total numbers of signals available in target application.	
<code>this</code>	<code>public xPCSignal Item[int signalIdx] {get;}</code> or <code>public xPCSignal Item[string blkPath] {get;}</code>	Return reference to xPCSignal object specified by its signal identifier (<i>signalIdx</i>) or signal name (<i>blkPath</i>). <i>signalIdx</i> is a 32-bit integer that identifies the signal.	xPCException — When problem occurs, query xPCException object Reason property. ArgumentNullException — <i>signalIdx</i> or

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

xPCStateLogger Class

Purpose Access to state log

Syntax `public class xPCStateLogger : xPCLog`

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

Syntax Language: C#

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

Properties

The xPCStateLogger class inherits its other properties from xPCLog Class.

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

Purpose Access xPCTargetPC class

Syntax `public xPCTargetPC()`

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

Syntax Language: C#

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

Constructor

Constructor	Description
xPCTargetPC	Construct xPCTargetPC object.

Methods

Method	Description
xPCTargetPC.Connect	Establish connection to target computer
xPCTargetPC.ConnectAsync	Asynchronous request for target computer connection
xPCTargetPC.Disconnect	Close connection
xPCTargetPC.DisconnectAsync	Begin asynchronous request to disconnect from target computer
xPCTargetPC.Dispose	Clean up resources
xPCTargetPC.Load	Load target application onto target computer
xPCTargetPC.LoadAsync	Begin asynchronous request for loading target application onto target computer
xPCTargetPC.Ping	Test communication between host and target computers
xPCTargetPC.Reboot	Reboot target computer
xPCTargetPC.RebootAsync	Begin asynchronous request to reboot target computer
xPCTargetPC.tcpPing	Determine TCP/IP accessibility of remote computer

xPCTargetPC Class

Method	Description
xPCTargetPC.Unload	Unload target application from target computer
xPCTargetPC.UnloadAsync	Begins asynchronous request to unload target application from target computer

Events

Event	Description
xPCTargetPC.ConnectCompleted	Event when asynchronous connect operation completes
xPCTargetPC.Connected	Event after establishing connection
xPCTargetPC.Connecting	Event before establishing connection
xPCTargetPC.DisconnectCompleted	Event when asynchronous disconnect operation completes
xPCTargetPC.Disconnected	Event after disconnect of established connection
xPCTargetPC.Disconnecting	Event before disconnection of established connection completes
xPCTargetPC.Disposed	Event after disposal of used resources
xPCTargetPC.LoadCompleted	Event when asynchronous load operation completes
xPCTargetPC.Loaded	Event when target application load operation completes
xPCTargetPC.Loading	Event before loading of target application on target computer
xPCTargetPC.RebootCompleted	Event when asynchronous reboot operation completes
xPCTargetPC.Rebooted	Event when target computer completes reboot
xPCTargetPC.Rebooting	Event before target computer reboots
xPCTargetPC.UnloadCompleted	Event when asynchronous target application unload operation completes
xPCTargetPC.Unloaded	Event when target application unloads from the target computer
xPCTargetPC.Unloading	Event before target application unloads from target computer

Properties

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

xPCTargetPC Class

Property	C# Declaration Syntax	Description	Exception
DLMFileName	public string DLMFileName {get; set;}	Get or set the full path to the DLM file name.	
Echo	public bool Echo {get; set;}	Get or set the target display on the target computer.	xPCException — When problem occurs, query xPCException object Reason property.
FileSystem	public xPCFileSystem FileSystem {get;}	Get a reference to an xPCFileSystem object that you can use to interface with the target file system. If no communication is established, the property returns a NULL object.	
HostTarget-Comm	public XPCProtocol HostTargetComm {get; set;}	Get or set the physical medium for communication. See xPCProtocol Enumerated Data Type.	
IsConnected	public bool IsConnected {get;}	Get connection status (established or not) to a remote target computer.	
IsConnecting-Busy	public bool IsConnectingBusy {get;}	Get ConnectAsync request status (in progress or not).	

Property	C# Declaration Syntax	Description	Exception
IsDisconnectingBusy	public bool IsDisconnectingBusy {get;}	Get whether a DisconnectAsync request is in progress.	
IsLoadingBusy	public bool IsLoadingBusy {get;}	Gets LoadAsync request status (in progress or not).	
IsRebooting-Busy	public bool IsRebootingBusy {get;}	Get RebootAsync request status (in progress or not).	
IsUnloading-Busy	public bool IsUnloadingBusy {get;}	Gets unLoadingAsync request status (in progress or not).	
RS232BaudRate	public XPCRS232BaudRate RS232Baudrate {get; set;}	Get or set baudrate for serial connection. See xPCRS232BaudRate Enumerated Data Type.	
RS232HostPort	public XPCRS232CommPort RS232HostPort {get; set;}	Get or set the serial COM port for connection on host computer. The xPC Target software automatically determines the COM port on the target computer. See xPCRS232Comport Enumerated Data Type.	

xPCTargetPC Class

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

Purpose Access to target scopes

Syntax `public class xPCTargetScope : xPCScope`

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

Syntax Language: C#

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

Methods

The xPCTargetScope class inherits methods from xPCScope Class.

Events

The xPCTargetScope class inherits events from xPCScope Class.

Properties

The xPCTargetScope class inherits its other properties from xPCScope Class.

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

xPCTargetScope Class

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

xPCTargetScopeCollection Class

Purpose Collection of xPCTargetScope objects

Syntax

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

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

Syntax Language: C#

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

Methods

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

xPCTargetScopeSignalCollection Class

Purpose Collection of xPCHostScopeSignal objects

Syntax

```
public class xPCTargetScopeSignalCollection : xPCScopeSignalCollection
```

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

Syntax Language: C#

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

Methods

Method	Description
xPCTargetScopeSignalCollection.Add	Adds an xPCTargetScopeSignal object
xPCTargetScopeSignalCollection.Refresh	Refreshes signals for associated target scopes on target computer

Properties

Property	C# Declaration Syntax	Description	Exception
Item	<pre>public xPCTgtScopeSignal Item[string blkpath] {get;}</pre>	<p>Get xPCTgtScopeSignal object from signal name (<i>blkpath</i>).</p> <p><i>blkpath</i> is the signal name that represents a signal object added to its parent xPCTargetScope object.</p> <p>This property returns the file scope signal</p>	xPCException — When problem occurs, query xPCException object Reason property.

xPCTargetScopeSignalCollection Class

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

xPCTETLogger Class

Purpose Access to 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	<code>public xPCDataLoggingObject DataLogObject {get;}</code>	Get TET data logging object.
IsEnabled	<code>public override bool IsEnabled {get;}</code>	Get whether to enable or disable logging. Overrides xPCLog.IsEnabled.

xPCTgtScopeSignal Class

Purpose Access to target scope signals

Syntax `public class xPCTgtScopeSignal : xPCScopeSignal`

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

Syntax Language: C#

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

Properties

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

xPCTimeLogger Class

Purpose Access to output log

Syntax `public class xPCTimeLogger : xPCLog`

Description **Namespace:** MathWorks.xPCTarget.FrameWork
Syntax Language: C#

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

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

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

Purpose Open file

Syntax `public xPCFileStream Open(xPCFileMode fileMode)`

Description **Class:** xPCFileInfo Class

Method

Syntax Language: C#

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

Exception

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

xPCFileInfo.OpenRead

Purpose Create read-only xPCFileStream object

Syntax `public xPCFileStream OpenRead()`

Description Class: xPCFileInfo Class

Method

Syntax Language: C#

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

Exception

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

Purpose Test communication between host and target computers

Syntax `public bool Ping()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

xPCFileStream.Read

Purpose Read block of bytes from stream and write data to buffer

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

Description Class: xPCFileStream Class

Method

Syntax Language: C#

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

Exception

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

Purpose Reboot target computer

Syntax `public void Reboot()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

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

xPCTargetPC.RebootAsync

Purpose Begin asynchronous request to reboot target computer

Syntax `public void RebootAsync()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

Purpose	Event when asynchronous reboot operation completes
Syntax	<code>public event RebootCompletedEventHandler RebootCompleted</code>
Description	<p>Class: xPCTargetPC Class</p> <p>Event</p> <p>Syntax Language: <i>C#</i></p> <p><code>public event RebootCompletedEventHandler RebootCompleted</code> occurs when asynchronous reboot operation completes.</p>

xPCTargetPC.Rebooted

Purpose Event after target computer completes reboot

Syntax `public event EventHandler Rebooted`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Rebooted` occurs after target computer reboot completes.

Purpose Event before target computer completes rebooting

Syntax `public event EventHandler Rebooting`

Description **Class:** xPCTargetPC Class

Event

Syntax Language: *C#*

`public event EventHandler Rebooting` occurs before target computer completes rebooting.

xPCFileScopeCollection.Refresh

Purpose Synchronize with file scopes on target computer

Syntax `public override void Refresh()`

Description **Class:** xPCFileScopeCollection Class

Method

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCFileScope>.Refresh().

Purpose Refresh state of object

Syntax `public void RefreshAll()`

Description **Class:** xPCScopes Class

Method

Syntax Language: C#

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

xPCDriveInfo.Refresh

Purpose Synchronize with file drives on target computer

Syntax `public void Refresh()`

Description Class: xPCDriveInfo Class

Method

Syntax Language: C#

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

xPCFileScopeSignalCollection.Refresh

Purpose Synchronize with signals for associated scope on target computer

Syntax `public override void Refresh()`

Description **Class:** xPCFileScopeSignalCollection Class

Method

Syntax Language: C#

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

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

Exception

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

xPCHostScopeCollection.Refresh

Purpose Refresh host scope object state

Syntax `public override void Refresh()`

Description **Class:** xPCHostScopeCollection Class

Method

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCHostScope>.Refresh().

Exception

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

xPCHostScopeSignalCollection.Refresh

Purpose Synchronize signals for associated host scopes on target computer

Syntax `public override void Refresh()`

Description **Class:** xPCHostScopeSignalCollection Class

Method

Syntax Language: C#

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

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

Exception

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

xPCParameters.Refresh

Purpose Refresh state of object

Syntax `public override void Refresh()`

Description **Class:** xPCParameters Class

Method

Syntax Language: C#

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

Purpose Refresh state of object

Syntax `public void Refresh()`

Description **Class:** xPCSignals Class

Method

Syntax Language: C#

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

xPCTargetScopeCollection.Refresh

Purpose Refresh target scope object state

Syntax `public override void Refresh()`

Description **Class:** xPCTargetScopeCollection Class

Method

Syntax Language: C#

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

Overrides xPCScopeCollection<xPCTargetScope>.Refresh().

xPCTargetScopeSignalCollection.Refresh

Purpose Synchronize signals for associated target scopes on target computer

Syntax `public override void Refresh()`

Description **Class:** xPCTargetScopeSignalCollection Class

Method

Syntax Language: C#

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

Overrides xPCScopeSignalCollection<xPCTgtScopeSignal>.Refresh().

Exception

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

xPCFileSystem.RemoveFile

Purpose Remove file name from target computer

Syntax `public void RemoveFile(string fileName)`

Description Class: xPCFileSystem Class

Method

Syntax Language: C#

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

Exception

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

Purpose Rename file

Syntax `public xPCFileInfo Rename(string newName)`

Description **Class:** xPCFileInfo **Class**

Method

Syntax Language: C#

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

Exception

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

xPCParameters.SaveParameterSet

Purpose Save parameter values of target application

Syntax `public void SaveParameterSet(string fileName)`

Description Class: xPCParameters Class

Method

Syntax Language: C#

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

Exception

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

SCDISPLAYMODE Enumerated Data Type

Purpose Target scope display mode values

Syntax `public enum SCDISPLAYMODE`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

SCFILEMODE Enumerated Data Type

Purpose Write mode values for when file allocation table entry is updated

Syntax `public enum SCFILEMODE`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

Purpose Event after scope start command completes

Syntax `public event EventHandler ScopeStarted`

Description **Class:** xPCScope Class

Event

Syntax Language: *C#*

`public event EventHandler ScopeStarted` occurs after scope start command completes.

xPCScope.ScopeStarting

Purpose Event before scope completes starting

Syntax `public event EventHandler ScopeStarting`

Description Class: xPCScope Class

Event

Syntax Language: C#

`public event EventHandler ScopeStarting` occurs before scope completes starting.

Purpose Event after scope completes manual stop command

Syntax `public event EventHandler ScopeStarting`

Description **Class:** xPCScope Class

Event

Syntax Language: C#

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

xPCScope.ScopeStopping

Purpose Event before scope completes manual stopping

Syntax `public event EventHandler ScopeStopping`

Description Class: xPCScope Class

Event

Syntax Language: C#

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

SCSTATUS Enumerated Data Type

Purpose Scope status values

Syntax `public enum SCSTATUS`

Description Enumerated Data Type

Syntax Language: C#

`public enum SCSTATUS` specifies scope status values.

Members

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

SCTRIGGERMODE Enumerated Data Type

Purpose Scope trigger mode values

Syntax `public enum SCTRIGGERMODE`

Description Enumerated Data Type

Syntax Language: C#

`public enum SCTRIGGERMODE` specifies scope trigger mode values.

Members

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

SCTRIGGERSLOPE Enumerated Data Type

Purpose Scope trigger slope values

Syntax `public enum SCTRIGGERSLOPE`

Description Enumerated Data Type

Syntax Language: C#

`public enum SCTRIGGERSLOPE` specifies scope trigger slope values.

Members

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

SCTYPE Enumerated Data Type

Purpose Scope type

Syntax `public enum SCTYPE`

Description Enumerated Data Type
Syntax Language: C#
`public enum SCTYPE` specifies scope type.

Members

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

xPCFileSystem.SetCurrentDirectory

Purpose Current folder

Syntax `public void SetCurrentDirectory(string path)`

Description Class: xPCFileSystem Class

Method

Syntax Language: C#

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

Exception

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

xPCParameter.SetParam

Purpose Change value of parameter

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

Description Class: xPCParameter Class

Method

Syntax Language: C#

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

Exception

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

Purpose Asynchronous request to set parameter values on target computer

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

Description **Class:** xPCParameter Class

Method

Syntax Language: C#

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

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCParameter.SetParamCompleted

Purpose Event when a set parameter operation completes

Description Class: xPCParameter Class

Event

Syntax Language: C#

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

Purpose Start target application simulation

Syntax `public void Start()`

Description **Class:** xPCApplication Class

Method

Syntax Language: C#

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

Exception

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

xPCFileScopeCollection.StartAll

Purpose Start all file scopes in one call

Syntax `public void StartAll()`

Description Class: xPCFileScopeCollection Class

Method

Syntax Language: C#

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

Purpose Start all host scopes in one call

Syntax `public void StartAll()`

Description **Class:** xPCHostScopeCollection Class

Method

Syntax Language: C#

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

Exception

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

xPCTargetScopeCollection.StartAll

Purpose Start all target scopes in one call

Syntax `public void StartAll()`

Description **Class:** xPCTargetScopeCollection Class

Method

Syntax Language: C#

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

Purpose Start scope

Syntax `public void Start()`

Description **Class:** xPCScope Class

Method

Syntax Language: C#

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

Exception

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

xPCApplication.Started

Purpose Event after simulation start command completes

Syntax `public event EventHandler Started`

Description Class: xPCApplication Class

Event

Syntax Language: *C#*

`public event EventHandler Started` occurs after target application start command completes.

Purpose	Event issued before simulation start command completes
Syntax	<code>public event EventHandler Starting</code>
Description	<p>Class: xPCApplication Class</p> <p>Event</p> <p>Syntax Language: C#</p> <p><code>public event EventHandler Starting</code> occurs before target application start command completes.</p>

xPCApplication.Stop

Purpose Stop target application simulation

Syntax `public void Stop()`

Description Class: xPCApplication Class

Method

Syntax Language: C#

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

Exception

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

Purpose Stop all file scopes in one call

Syntax `public void StartAll()`

Description **Class:** xPCFileScopeCollection Class

Method

Syntax Language: C#

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

xPCHostScopeCollection.StopAll

Purpose Stop all host scopes in one call

Syntax `public void StartAll()`

Description Class: xPCHostScopeCollection Class

Method

Syntax Language: C#

`public void StartAll()` 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.

Purpose Stop all target scopes in one call

Syntax `public void StartAll()`

Description **Class:** xPCTargetScopeCollection Class

Method

Syntax Language: C#

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

xPCScope.Stop

Purpose Stop scope

Syntax `public void Stop()`

Description Class: xPCScope Class

Method

Syntax Language: C#

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

Exception

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

Purpose

Event after target application stop command completes

Syntax

```
public event EventHandler Stopped
```

Description

Class: xPCApplication Class

Event

Syntax Language: C#

`public event EventHandler Stopped` occurs after target application stop command completes.

xPCApplication.Stopping

Purpose Event before target application stop command completes

Syntax `public event EventHandler Stopping`

Description Class: xPCApplication Class

Event

Syntax Language: *C#*

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

Purpose Determine TCP/IP accessibility of remote computer

Syntax `public bool tcpPing()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: *C#*

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

xPCScope.Trigger

Purpose Software-trigger start of data acquisition for scope

Syntax `public void Trigger()`

Description Class: xPCScope Class

Method

Syntax Language: C#

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

Exception

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

Purpose Status of get signal value at moment of request

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

Description **Class:** xPCSignal Class

Method

Syntax Language: *C#*

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

xPCTargetPC.Unload

Purpose Unload target application from target computer

Syntax `public void Unload()`

Description Class: xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

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

Purpose Begin asynchronous request to unload target application from target computer

Syntax `public void UnloadAsync()`

Description **Class:** xPCTargetPC Class

Method

Syntax Language: C#

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

Exception

Exception	Condition
InvalidOperationException	When another thread uses this method.

xPCTargetPC.UnloadCompleted

Purpose	Event when asynchronous target application unload operation completes
Syntax	<code>public event UnloadCompletedEventHandler UnloadCompleted</code>
Description	<p>Class: xPCTargetPC Class</p> <p>Event</p> <p>Syntax Language: C#</p> <p><code>public event UnloadCompletedEventHandler UnloadCompleted</code> occurs when asynchronous target application unload operation completes.</p>

Purpose	Event after target application unload from the target computer completes
Syntax	<code>public event EventHandler Unloaded</code>
Description	<p>Class: xPCTargetPC Class</p> <p>Event</p> <p>Syntax Language: C#</p> <p><code>public event EventHandler Unloaded</code> occurs after target application unload from the target computer completes.</p>

xPCTargetPC.Unloading

Purpose Event before target application unload from target computer completes

Syntax `public event EventHandler Unloading`

Description Class: xPCTargetPC Class

Event

Syntax Language: *C#*

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

Purpose Write block of bytes to file stream

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

Description **Class:** xPCFileStream Class

Method

Syntax Language: C#

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

Exception

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

xPCFileStream.WriteByte

Purpose Write byte to current position in file stream

Syntax `public void WriteByte(byte value)`

Description Class: xPCFileStream Class

Method

Syntax Language: C#

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

Exception

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

xPCAppStatus Enumerated Data Type

Purpose Target application status return values

Syntax `public enum xPCAppStatus`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

xPCDirectoryInfo

Purpose Construct new instance of the xPCDirectoryInfo class on specified path

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

Description Class: xPCDirectoryInfo Class

Constructor

Syntax Language: C#

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

Exception

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

Purpose Construct new instance of xPCDriveInfo class

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

Description **Class:** xPCDriveInfo Class

Constructor

Syntax Language: C#

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

Exception

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

xPCException

Purpose Construct new instance of xPCException class

Syntax

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

Description Class: xPCException Class

Constructor

Syntax Language: C#

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

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

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

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

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

xPCExceptionReason Enumerated Data Type

Purpose Exception reasons

Syntax `public enum xPCExceptionReason`

Description Enumerated Data Type

Syntax Language: *C#*

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

xPCFileInfo

Purpose Construct new instance of xPCFileInfo class

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

Description Class: xPCFileInfo Class

Constructor

Syntax Language: C#

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

Exception

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

xPCFileMode Enumerated Data Type

Purpose Open file with permissions

Syntax `public enum xPCFileMode`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

xPCFileStream

Purpose Construct new instance of xPCFileStream class

Syntax

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

Description Class: xPCFileStream Class

Method

Syntax Language: C#

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

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

Exception

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

Purpose Construct new instance of xPCFileSystemInfo class

Syntax `public xPCFileSystemInfo(xPCTargetPC tgt)`

Description **Class:** xPCFileSystemInfo Class

Constructor

Syntax Language: C#

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

xPCLogMode Enumerated Data Type

Purpose Specify log mode values

Syntax `public enum xPCLogMode`

Description Enumerated Data Type

Syntax Language: C#

`public enum xPCLogMode` specifies log mode values.

Members

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

xPCLogType Enumerated Data Type

Purpose Logging type values

Syntax `public enum xPCLogType`

Description Enumerated Data Type

Syntax Language: C#

`public enum xPCLogType` specifies logging type values.

Members

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

xPCProtocol Enumerated Data Type

Purpose Host computer and target computer communication medium

Syntax `public enum XPCProtocol`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

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

xPCRS232BaudRate Enumerated Data Type

Purpose Serial communication baud rate

Syntax `public enum XPCRS232BaudRate`

Description Enumerated Data Type

Syntax Language: C#

`public enum XPCRS232BaudRate` specifies serial communication baud rate

Members

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

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

xPCRS232Comport Enumerated Data Type

Purpose Serial communication port

Syntax `public enum XPCRS232CommPort`

Description Enumerated Data Type

Syntax Language: C#

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

Members

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

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

Purpose Construct new instance of xPCTargetPC class

Syntax `public xPCTargetPC()`

Description **Class:** xPCTargetPC Class

Constructor

Syntax Language: C#

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

xPC Target API Reference for C

- “C API Functions” on page 7-2
- “C API Error Messages” on page 7-10
- “C API Structures and Functions — Alphabetical List” on page 7-14

C API Functions

In this section...
“Target Computers” on page 7-2
“Target Applications” on page 7-3
“Scopes” on page 7-4
“Parameters” on page 7-6
“Signals” on page 7-7
“Data Logs” on page 7-7
“File Systems” on page 7-8
“Errors” on page 7-9

Target Computers

xPCCloseConnection	Close RS-232 or TCP/IP communication connection
xPCClosePort	Close RS-232 or TCP/IP communication connection
xPCDeRegisterTarget	Delete target communication properties from xPC Target API library
xPCFreeAPI	Unload xPC Target DLL
xPCGetEcho	Return display mode for target message window
xPCGetLoadTimeOut	Return timeout value for communication between host computer and target computer
xPCInitAPI	Initialize xPC Target DLL
xPCIsAppRunning	Return target application running status
xPCOpenConnection	Open connection to target computer

xPCOpenSerialPort	Open RS-232 connection to xPC Target system
xPCOpenTcpIpPort	Open TCP/IP connection to xPC Target system
xPCReboot	Reboot target computer
xPCRegisterTarget	Register target with xPC Target API library
xPCReOpenPort	Reopen communication channel
xPCSetEcho	Turn message display on or off
xPCSetLoadTimeOut	Change initialization timeout value between host computer and target computer
xPCTargetPing	Ping target computer
xPCUnloadApp	Unload target application

Target Applications

xPCAverageTET	Return average task execution time
xPCGetAPIVersion	Get version number of xPC Target API
xPCGetAppName	Return target application name
xPCGetExecTime	Return target application execution time
xPCGetSampleTime	Return target application sample time
xPCGetSessionTime	Return length of time xPC Target kernel has been running
xPCGetStopTime	Return stop time
xPCGetTargetVersion	Get xPC Target kernel version
xPCIsOverloaded	Return target computer overload status

xPCLoadParamSet	Restore parameter values
xPCMaximumTET	Copy maximum task execution time to array
xPCMinimumTET	Copy minimum task execution time to array
xPCSaveParamSet	Save parameter values of target application
xPCSetSampleTime	Change target application sample time
xPCSetStopTime	Change target application stop time
xPCStartApp	Start target application
xPCStopApp	Stop target application

Scopes

scopedata	Type definition for scope data structure
xPCAddScope	Create new scope
xPCGetNumScopes	Return number of scopes added to target application
xPCGetNumScSignals	Returns number of signals added to specific scope
xPCGetScope	Get and copy scope data to structure
xPCGetScopeList	Get and copy list of scope numbers
xPCGetScopes	Get and copy list of scope numbers
xPCIsScFinished	Return data acquisition status for scope
xPCRemScope	Remove scope
xPCScAddSignal	Add signal to scope
xPCScGetAutoRestart	Scope autorestart status

xPCScGetData	Copy scope data to array
xPCScGetDecimation	Return decimation of scope
xPCScGetNumPrePostSamples	Get number of pre- or post-triggering samples before triggering scope
xPCScGetNumSamples	Get number of samples in one data acquisition cycle
xPCScGetNumSignals	Get number of signals in scope
xPCScGetSignalList	Copy list of signals to array
xPCScGetSignals	Copy list of signals to array
xPCScGetStartTime	Get start time for last data acquisition cycle
xPCScGetState	Get state of scope
xPCScGetTriggerLevel	Get trigger level for scope
xPCScGetTriggerMode	Get trigger mode for scope
xPCScGetTriggerScope	Get trigger scope
xPCScGetTriggerScopeSample	Get sample number for triggering scope
xPCScGetTriggerSignal	Get trigger signal for scope
xPCScGetTriggerSlope	Get trigger slope for scope
xPCScGetType	Get type of scope
xPCScRemSignal	Remove signal from scope
xPCScSetAutoRestart	Scope autorestart status
xPCScSetDecimation	Set decimation of scope
xPCScSetNumPrePostSamples	Set number of pre- or posttriggering samples before triggering scope
xPCScSetNumSamples	Set number of samples in one data acquisition cycle
xPCScSetTriggerLevel	Set trigger level for scope
xPCScSetTriggerMode	Set trigger mode of scope

xPCScSetTriggerScope	Select scope to trigger another scope
xPCScSetTriggerScopeSample	Set sample number for triggering scope
xPCScSetTriggerSignal	Select signal to trigger scope
xPCScSetTriggerSlope	Set slope of signal that triggers scope
xPCScSoftwareTrigger	Set software trigger of scope
xPCScStart	Start data acquisition for scope
xPCScStop	Stop data acquisition for scope
xPCSetScope	Set properties of scope
xPCTgScGetGrid	Get status of grid line for particular scope
xPCTgScGetMode	Get scope mode for displaying signals
xPCTgScGetViewMode	Get view mode for target computer display
xPCTgScGetYLimits	Copy y-axis limits for scope to array
xPCTgScSetGrid	Set grid mode for scope
xPCTgScSetMode	Set display mode for scope
xPCTgScSetViewMode	Set view mode for scope
xPCTgScSetYLimits	Set y-axis limits for scope

Parameters

xPCGetNumParams	Return number of tunable parameters
xPCGetParam	Get parameter value and copy it to array
xPCGetParamDims	Get row and column dimensions of parameter
xPCGetParamIdx	Return parameter index

xPCGetParamName	Get name of parameter
xPCSetParam	Change value of parameter

Signals

xPCGetNumSignals	Return number of signals
xPCGetSigIdxfromLabel	Return array of signal indices
xPCGetSigLabelWidth	Return number of elements in signal
xPCGetSignal	Return value of signal
xPCGetSignalIdx	Return index for signal
xPCGetSignalLabel	Copy label of signal to character array
xPCGetSignalName	Copy name of signal to character array
xPCGetSignals	Return vector of signal values
xPCGetSignalWidth	Return width of signal

Data Logs

lgmode	Type definition for logging options structure
xPCGetLogMode	Return logging mode and increment value for target application
xPCGetNumOutputs	Return number of outputs
xPCGetNumStates	Return number of states
xPCGetOutputLog	Copy output log data to array
xPCGetStateLog	Copy state log values to array
xPCGetTETLog	Copy TET log to array
xPCGetTimeLog	Copy time log to array

xPCMaxLogSamples	Return maximum number of samples that can be in log buffer
xPCNumLogSamples	Return number of samples in log buffer
xPCNumLogWraps	Return number of times log buffer wraps
xPCSetLogMode	Set logging mode and increment value of scope

File Systems

dirStruct	Type definition for file system folder information structure
diskinfo	Type definition for file system disk information structure
fileinfo	Type definition for file information structure
xPCFSCD	Change current folder on target computer to specified path
xPCFSCloseFile	Close file on target computer
xPCFSDir	Get contents of specified folder on target computer
xPCFSDirItems	Get contents of specified folder on target computer
xPCFSDirSize	Return size of specified folder listing on target computer
xPCFSDirStructSize	Get number of items in folder
xPCFSDiskInfo	Information about target computer file system
xPCFSFileInfo	Return information for open file on target computer
xPCFSGetFileSize	Return size of file on target computer

xPCFSGetPWD	Get current folder of target computer
xPCFSMKDIR	Create new folder on target computer
xPCFSOpenFile	Open file on target computer
xPCFSReadFile	Read open file on target computer
xPCFSRemoveFile	Remove file from target computer
xPCFSRMDIR	Remove folder from target computer
xPCFSScGetFilename	Get name of file for scope
xPCFSScGetWriteMode	Get write mode of file for scope
xPCFSScGetWriteSize	Get block write size of data chunks
xPCFSScSetFilename	Specify name for file to contain signal data
xPCFSScSetWriteMode	Specify when file allocation table entry is updated
xPCFSScSetWriteSize	Specify that memory buffer collect data in multiples of write size
xPCFSWriteFile	Write to file on target computer

Errors

xPCErrorMsg	Return text description for error message
xPCFSError	Get text description for error number on target computer file system
xPCGetLastError	Return constant of last error
xPCSetLastError	Set last error to specific string constant

C API Error Messages

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

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

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

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

Message	Description
ETGTMEMALLOC	Target memory allocation failed
ETIMELOGDISABLED	Time Logging is disabled
ETOOMANYSAMPLES	Too Many Samples requested
ETOOMANYSCOPES	Too many scopes are present
ETOOMANYSIGNALS	Too many signals in Scope
EUNLOADAPPFIRST	Unload the application first
EUSEDYNSCOPE	Use DYNAMIC_SCOPE flag at compile time
EWRITEFILE	LoadDLM: WriteFile Error
EWSINIT	WINSOCK: Initialization Error
EWSNOTREADY	Winsock not ready

C API Structures and Functions – Alphabetical List

Purpose Type definition for file system folder information structure

Syntax

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

Fields

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

dirStruct

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

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

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

See Also API function `xPCFSDirItems`

Purpose Type definition for file system disk information structure

Syntax

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

Fields

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

<i>FATType</i>	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of <i>Files</i> .

<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to <code>FreeClusters</code> .

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

See Also API function `xPCFSDiskInfo`

fileinfo

Purpose Type definition for file information structure

Syntax

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

Fields

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

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

See Also xPCFSFileInfo

Purpose	Type definition for logging options structure				
Syntax	<pre>typedef struct { int mode; double incrementvalue; } lgmode;</pre>				
Fields	<table><tr><td><i>mode</i></td><td>This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.</td></tr><tr><td><i>incrementvalue</i></td><td>If you set <i>mode</i> 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 <i>incrementvalue</i>. If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.</td></tr></table>	<i>mode</i>	This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.	<i>incrementvalue</i>	If you set <i>mode</i> 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 <i>incrementvalue</i> . If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.
<i>mode</i>	This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.				
<i>incrementvalue</i>	If you set <i>mode</i> 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 <i>incrementvalue</i> . If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.				
Description	The lgmode structure specifies data logging options. The <i>mode</i> variable accepts either the numeric values 0 or 1 or their equivalent constants LGMOD_TIME or LGMOD_VALUE from xpcapiconst.h.				
See Also	API functions xPCSetLogMode, xPCGetLogMode				

scopedata

Purpose Type definition for scope data structure

Syntax

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

Fields

<i>number</i>	The scope number.
<i>type</i>	Determines whether the scope is displayed on the host computer or on the target computer. Values are one of the following: 1 Host 2 Target
<i>state</i>	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
<i>signals</i>		List of signal indices from the target object to display on the scope.
<i>numsamples</i>		Number of contiguous samples captured during the acquisition of a data package.
<i>decimation</i>		A number, N, meaning every Nth sample is acquired in a scope window.
<i>triggermode</i>		Trigger mode for a scope. Values are one of the following:
	0	FreeRun (default)
	1	Software
	2	Signal
	3	Scope
<i>numprepostsamples</i>		If this value is less than 0, this is the number of samples to be saved before a trigger event. If this value is greater than 0, this is the number of samples to skip after the trigger event before data acquisition begins.
<i>triggersignal</i>		If <i>triggermode</i> is 2 (Signal), identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.
<i>triggerscope</i>		If <i>triggermode</i> is 3 (Scope), identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.
<i>triggerscopesample</i>		If <i>triggermode</i> is 3 (Scope), specifies the number of samples to be acquired by the triggering scope before triggering a second scope. This must be a nonnegative value.

<i>triggerlevel</i>	If <i>triggermode</i> is 2 (Signal), indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with either a rising or falling signal.
<i>triggerslope</i>	If <i>triggermode</i> is 2 (Signal), indicates whether the trigger is on a rising or falling signal. Values are: 0 Either rising or falling (default) 1 Rising 2 Falling

Description

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

See Also

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

Purpose Create new scope

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

Arguments

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

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

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

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

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

See Also API functions `xPCScAddSignal`, `xPCScRemSignal`, `xPCRemScope`, `xPCSetScope`, `xPCGetScope`, `xPCGetScopes`

Target object method `xpctarget.xpc.addscope`

xPCAverageTET

Purpose	Return average task execution time
Prototype	<code>double xPCAverageTET(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCAverageTET</code> function returns the average task execution time (TET) for the target application.
Description	The <code>xPCAverageTET</code> function returns the TET for the target application. You can use this function when the target application is running or when it is stopped.
See Also	API functions <code>xPCMaximumTET</code> , <code>xPCMinimumTET</code> Target object property <code>AvgTET</code>

Purpose Close RS-232 or TCP/IP communication connection

Prototype `void xPCCloseConnection(int port);`

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

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

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

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

xPCClosePort

Purpose Close RS-232 or TCP/IP communication connection

Prototype `void xPCClosePort(int port);`

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

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

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

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

Target object method `xpctarget.xpc.close`

Purpose Delete target communication properties from xPC Target API library

Prototype `void xPCDeRegisterTarget(int port);`

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

Description The `xPCDeRegisterTarget` function causes the xPC Target API library to completely “forget” about the target communication properties. It works similarly to `xPCClosePort`, but does not close the connection to the target machine. Before calling this function, you must first call the function `xPCCloseConnection` to close the connection to the target machine. The combination of calling the `xPCCloseConnection` and `xPCDeRegisterTarget` functions has the same result as calling `xPCClosePort`.

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

xPCErrorMsg

Purpose Return text description for error message

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

Arguments

error_number Enter the constant of an error.

error_message The `xPCErrorMsg` function copies the error message string into the buffer pointed to by *error_message*. *error_message* is then returned. You can later use *error_message* in a function such as `printf`.

If *error_message* is `NULL`, the `xPCErrorMsg` function returns a pointer to a statically allocated string.

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

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

See Also API functions `xPCSetLastError`, `xPCGetLastError`

Purpose Unload xPC Target DLL

Prototype `void xPCFreeAPI(void);`

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

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

xPCFSCD

Purpose Change current folder on target computer to specified path

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dir</i>	Enter the path on the target computer to change to.

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

See Also

- API function `xPCFSGetPWD`
- File object method `xpctarget.fsbase.cd`

Purpose Close file on target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

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

See Also API functions `xPCFSOpenFile`, `xPCFSReadFile`, `xPCFSWriteFile`
File object method `xpctarget.fs.fclose`

xPCFSDir

Purpose Get contents of specified folder on target computer

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

Arguments

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

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

See Also API function `xPCFSDirSize`
File object method `xpctarget.fsbase.dir`

Purpose

Get contents of specified folder on target computer

Prototype

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

Arguments

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

path Enter the path on the target computer.

dirs Enter the structure to contain the contents of the folder.

numDirItems Enter the number of items in the folder.

Description

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

See Also

API functions xPCFSDirStructSize, dirStruct
File object method xpctarget.fsbasedir

xPCFSDirSize

Purpose Return size of specified folder listing on target computer

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

Arguments

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

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

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

See Also API function xPCFSDirItems
File object method `xpctarget.fsbase.dir`

Purpose Get number of items in folder

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the folder path on the target computer.

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

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

See Also

- API function `xPCFSDir`
- File object method `xpctarget.fsbase.dir`

xPCFSDiskInfo

Purpose Information about target computer file system

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>driveletter</i>	Enter the drive letter of the file system for which you want information.

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

See Also API structure `xpctarget.fs.diskinfo`

Purpose	Return information for open file on target computer				
Prototype	<code>fileinfo xPCFSFileInfo(int <i>port</i>, int <i>fileHandle</i>);</code>				
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.				
Description	The <code>xPCFSFileInfo</code> function returns information about the specified open file, <code>filehandle</code> , in a structure of type <code>fileinfo</code> .				
See Also	Structure <code>xpctarget.fs.fileinfo</code>				

xPCFSError

Purpose Get text description for error number on target computer file system

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

Arguments

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

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

Purpose Return size of file on target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

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

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

See Also API functions `xPCFSOpenFile`, `xPCFSReadFile`
File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

xPCFSGetPWD

Purpose Get current folder of target computer

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

Arguments

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

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

See Also File object method `xpctarget.fsbase.pwd`

Purpose Create new folder on target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of the folder to create on the target computer.

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

See Also

- API function `xPCFSGetPWD`
- File object method `xpctarget.fsbased.mkdir`

xPCFSOpenFile

Purpose Open file on target computer

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

Arguments

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

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

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

See Also API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSReadFile`, `xPCFSWriteFile`
File object methods `xpctarget.fs.fclose`, `xpctarget.fs.filetable`, `xpctarget.fs.fopen`, `xpctarget.fs.fread`, `xpctarget.fs.fwrite`

Purpose Read open file on target computer

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

Arguments

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

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

See Also API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSWriteFile`
File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

xPCFSRemoveFile

Purpose Remove file from target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of a file on the target computer.

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

See Also File object method `xpctarget.fs.removefile`

Purpose Remove folder from target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of a folder on the target computer.

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

See Also File object method `xpctarget.fsbase.rmdir`

xPCFSScGetFilename

Purpose Get name of file for scope

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

Arguments

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

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

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

See Also API function `xPCFSScSetFilename`
Scope object property `Filename`

Purpose Get write mode of file for scope

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

Arguments

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

Return Returns the number indicating the write mode. Values are

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

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

See Also API function `xPCFSScSetWriteMode`
Scope object property `Mode`

xPCFSScGetWriteSize

Purpose Get block write size of data chunks

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

Arguments

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

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

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

See Also API function `xPCFSScSetWriteSize`
Scope object property `WriteSize`

Purpose Specify name for file to contain signal data

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

Arguments

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

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

See Also API function `xPCFSScGetFilename`
Scope object property `Filename`

xPCFSScSetWriteMode

Purpose Specify when file allocation table entry is updated

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeMode</i>	Enter an integer for the write mode: 0 Enables lazy write mode 1 Enables commit write mode

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

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

See Also API function `xPCFSScGetWriteMode`
Scope object property `Mode`

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

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

Arguments

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

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

See Also API function `xPCFSScGetWriteSize`
Scope object property `WriteSize`

xPCFSWriteFile

Purpose Write to file on target computer

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

Arguments

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

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

See Also API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSReadFile`

Purpose	Get version number of xPC Target API
Prototype	<code>const char *xPCGetAPIVersion(void);</code>
Return	The <code>xPCGetApiVersion</code> function returns a string with the version number of the xPC Target kernel on the target computer.
Description	The <code>xPCGetApiVersion</code> function returns a string with the version number of the xPC Target kernel on the target computer. The string is a constant string within the API DLL. Do not modify this string.
See Also	API function <code>xPCGetTargetVersion</code>

xPCGetAppName

Purpose Return target application name

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>model_name</i>	The <code>xPCGetAppName</code> function copies the target application name string into the buffer pointed to by <i>model_name</i> . <i>model_name</i> is then returned. You can later use <i>model_name</i> in a function such as <code>printf</code> . Note that the maximum size of the buffer is 256 bytes. To reserve enough space for the application name string, allocate a buffer of size 256 bytes.

Return The `xPCGetAppName` function returns a string with the name of the target application.

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

Examples Allocate 256 bytes for the buffer `appname`.

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

See Also API function `xPCIsAppRunning`
Target object property `Application`

Purpose Return display mode for target message window

Prototype `int xPCGetEcho(int port);`

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

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

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

0 Display is off.

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

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

See Also API function `xPCSetEcho`

xPCGetExecTime

Purpose	Return target application execution time
Prototype	<code>double xPCGetExecTime(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetExecTime</code> function returns the current execution time for a target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetExecTime</code> function returns the current execution time for the running target application. If the target application is stopped, the value is the last running time when the target application was stopped. If the target application is running, the value is the current running time.
See Also	API functions <code>xPCSetStopTime</code> , <code>xPCGetStopTime</code> Target object property <code>ExecTime</code>

Purpose	Return constant of last error
Prototype	<code>int xPCGetLastError(void);</code>
Return	The xPCGetLastError function returns the error constant for the last reported error. If the function did not detect an error, it returns 0.
Description	The xPCGetLastError function returns the constant of the last reported error by another API function. This value is reset every time you call a new function. Therefore, you should check this constant value immediately after a call to an API function. For a list of error constants and messages, see “C API Error Messages”.
See Also	API functions xPCErrorMsg, xPCSetLastError

xPCGetLoadTimeOut

Purpose Return timeout value for communication between host computer and target computer

Prototype `int xPCGetLoadTimeOut(int port);`

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

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

Description The `xPCGetLoadTimeOut` function returns the number of seconds allowed for the communication between the host computer and the target application. When an xPC Target API function initiates communication between the host computer and target computer, it waits for a certain amount of time before checking to see if the communication is complete. In the case where communication with the target computer is not complete, the function returns a timeout error.

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

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

See Also

API functions xPCLoadApp, xPCSetLoadTimeOut

xPCUnloadApp

“Increase the Time for Downloads”

xPCGetLogMode

Purpose	Return logging mode and increment value for target application
Prototype	<code>lgmode xPCGetLogMode(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetLogMode</code> function returns the logging mode in the <code>lgmode</code> structure. If the logging mode is 1 (<code>LGMOD_VALUE</code>), this function also returns an increment value in the <code>lgmode</code> structure. If an error occurs, this function returns -1.
Description	The <code>xPCGetLogMode</code> function gets the logging mode and increment value for the current target application. The increment (difference in amplitude) value is measured between logged data points. A data point is logged only when an output signal or a state changes by the increment value.
See Also	API function <code>xPCSetLogMode</code> API structure <code>lgmode</code>

Purpose	Return number of outputs
Prototype	<code>int xPCGetNumOutputs(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetNumOutputs</code> function returns the number of outputs in the current target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetNumOutputs</code> function returns the number of outputs in the target application. The number of outputs equals the sum of the input signal widths of the output blocks at the root level of the Simulink model.
See Also	API functions <code>xPCGetOutputLog</code> , <code>xPCGetNumStates</code> , <code>xPCGetStateLog</code>

xPCGetNumParams

Purpose	Return number of tunable parameters
Prototype	<code>int xPCGetNumParams(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetNumParams</code> function returns the number of tunable parameters in the target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetNumParams</code> function returns the number of tunable parameters in the target application. Use this function to see how many parameters you can get or modify.
See Also	API functions <code>xPCGetParamIdx</code> , <code>xPCSetParam</code> , <code>xPCGetParam</code> , <code>xPCGetParamName</code> , <code>xPCGetParamDims</code> Target object property <code>NumParameters</code>

Purpose	Return number of scopes added to target application
Prototype	<code>int xPCGetNumScopes(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetNumScopes</code> function returns the number of scopes that have been added to the target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetNumScopes</code> function returns the number of scopes that have been added to the target application.

xPCGetNumScSignals

Purpose Returns number of signals added to specific scope

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

Arguments

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

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

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

Purpose	Return number of signals
Prototype	<code>int xPCGetNumSignals(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetNumSignals</code> function returns the number of signals in the target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetNumSignals</code> function returns the total number of signals in the target application that can be monitored from the host. Use this function to see how many signals you can monitor.
See Also	API functions <code>xPCGetSignalIdx</code> , <code>xPCGetSignal</code> , <code>xPCGetSignals</code> , <code>xPCGetSignalName</code> , <code>xPCGetSignalWidth</code> Target object property <code>NumSignals</code>

xPCGetNumStates

Purpose	Return number of states
Prototype	<code>int xPCGetNumStates(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetNumStates</code> function returns the number of states in the target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetNumStates</code> function returns the number of states in the target application.
See Also	API functions <code>xPCGetStateLog</code> , <code>xPCGetNumOutputs</code> , <code>xPCGetOutputLog</code> Target object property <code>StateLog</code>

Purpose Copy output log data to array

Prototype

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

Arguments

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

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

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

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

xPCGetOutputLog

See Also

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

Target object method `xpctarget.xpc.getlog`

Target object property `OutputLog`

Purpose Get parameter value and copy it to array

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

Arguments

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

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

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

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

Target object method `xpctarget.xpc.getparamid`

Target object properties `ShowParameters`, `Parameters`

xPCGetParamDims

Purpose Get row and column dimensions of parameter

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Parameter index.
<i>dimension</i>	Dimensions (row, column) of a parameter.

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

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

See Also API functions `xPCGetParamIdx`, `xPCGetParamName`, `xPCSetParam`, `xPCGetParam`, `xPCGetNumParams`

Target object method `xpctarget.xpc.getparamid`

Target object properties `ShowParameters`, `Parameters`

Purpose

Return parameter index

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API functions `xPCGetParamDims`, `xPCGetParamName`, `xPCGetParam`
Target object method `xpctarget.xpc.getparamid`
Target object properties `ShowParameters`, `Parameters`

xPCGetParamName

Purpose Get name of parameter

Prototype

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

Arguments

<i>port</i>	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
<i>paramIdx</i>	Enter a parameter index.
<i>blockName</i>	String with the full block path generated by Simulink Coder.
<i>paramName</i>	Name of a parameter for a specific block.

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

See Also API functions xPCGetParam, xPCGetParamDims, xPCGetParamIdx
Target object properties ShowParameters, Parameters

Purpose	Return target application sample time
Prototype	<code>double xPCGetSampleTime(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCGetSampleTime</code> function returns the sample time, in seconds, of the target application. If the function detects an error, it returns -1.
Description	The <code>xPCGetSampleTime</code> function returns the sample time, in seconds, of the target application. You can get the error by using the function <code>xPCGetLastError</code> .
See Also	API function <code>xPCSetSampleTime</code> Target object property <code>SampleTime</code>

xPCGetScope

Purpose	Get and copy scope data to structure				
Prototype	<code>scopedata xPCGetScope(int <i>port</i>, int <i>scNum</i>);</code>				
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>scNum</i></td><td>Enter the scope number.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>scNum</i>	Enter the scope number.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>scNum</i>	Enter the scope number.				
Return	The <code>xPCGetScope</code> function returns a structure of type <code>scopedata</code> .				
Description	The <code>xPCGetScope</code> function gets properties of a scope with <i>scNum</i> and copies the properties into a structure with type <code>scopedata</code> . You can use this function in conjunction with <code>xPCSetScope</code> to change several properties of a scope at one time. See <code>scopedata</code> for a list of properties. Use the <code>xPCGetScope</code> function to get the scope number.				
See Also	API functions <code>xPCSetScope</code> , <code>scopedata</code> Target object method <code>xpctarget.xpc.getscope</code>				

Purpose Get and copy list of scope numbers

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

Arguments

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

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

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

xPCGetScopes

Purpose Get and copy list of scope numbers

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers and terminated by -1.

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

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

See Also API functions `xPCSetScope`, `xPCGetScope`, `xPCScGetSignals`
Target object property `Scopes`

Purpose	Return length of time xPC Target kernel has been running		
Prototype	<code>double xPCGetSessionTime(int <i>port</i>);</code>		
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .		
Return	The <code>xPCGetSessionTime</code> function returns the amount of time in seconds that the xPC Target kernel has been running on the target computer. If the function detects an error, it returns -1.		
Description	The <code>xPCGetSessionTime</code> function returns, as a double, the amount of time in seconds that the xPC Target kernel has been running. This value is also the time that has elapsed since you last booted the target computer.		

xPCGetSignal

Purpose Return value of signal

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigNum</i>	Enter a signal number.

Return The `xPCGetSignal` function returns the current value of signal *sigNum*. If the function detects an error, it returns -1.

Description The `xPCGetSignal` function returns the current value of a signal. For vector signals, use `xPCGetSignals` rather than call this function multiple times. Use the `xPCGetSignalIdx` function to get the signal number.

See Also API function `xPCGetSignals`
Target object properties `ShowSignals`, `Signals`

Purpose	Return index for signal				
Prototype	<code>int xPCGetSignalIdx(int <i>port</i>, const char *<i>sigName</i>);</code>				
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>sigName</i></td><td>Enter a signal name.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>sigName</i>	Enter a signal name.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>sigName</i>	Enter a signal name.				
Return	The <code>xPCGetSignalIdx</code> function returns the index for the signal with name <i>sigName</i> . If the function detects an error, it returns -1.				
Description	The <code>xPCGetSignalIdx</code> function returns the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file <code>model_namebio.m</code> in the generated code, where <i>model_name</i> is the name of the model. The creator of the application should already know the signal name.				
See Also	API functions <code>xPCGetSignalName</code> , <code>xPCGetSignalWidth</code> , <code>xPCGetSignal</code> , <code>xPCGetSignals</code> Target object method <code>xpctarget.xpc.getsignalid</code>				

xPCGetSigIdxfromLabel

Purpose Return array of signal indices

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigLabel</i>	String with the name of a signal label.
<i>sigIds</i>	Return array of signal indices.

Return If `xPCGetSigIdxfromLabel` finds a signal, it fills an array *sigIds* with signal indices and returns 0. If it finds no signal, it returns -1.

Description The `xPCGetSigIdxfromLabel` function returns in *sigIds* the array of signal indices for signal *sigName*. This function assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

sigIds must be large enough to contain the array of indices. You can use the `xPCGetSigLabelWidth` function to get the required amount of memory to be allocated by the *sigIds* array.

See Also API functions `xPCGetSignalLabel`, `xPCGetSigLabelWidth`

Purpose Copy label of signal to character array

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

Arguments

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

Return The `xPCGetSignalLabel` function returns the label of the signal.

Description The `xPCGetSignalLabel` function copies and returns the signal label, including the block path, of a signal with *sigIdx*. The result is stored in *sigLabel*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigLabel* is unchanged. The function returns *sigLabel*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index. Signal labels must be unique.

This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

See Also API functions `xPCGetSigIdxfromLabel`, `xPCGetSigLabelWidth`

xPCGetSigLabelWidth

Purpose Return number of elements in signal

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigName</i>	String with the name of a signal.

Return The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. If the function detects an error, it returns -1.

Description The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. This function assumes that you have labeled the signal for which you request the elements (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

See Also API functions `xPCGetSigIdxfromLabel`, `xPCGetSignalLabel`

Purpose Copy name of signal to character array

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter a signal index.
<i>sigName</i>	String with the name of a signal.

Return The `xPCGetSignalName` function returns the name of the signal.

Description The `xPCGetSignalName` function copies and returns the signal name, including the block path, of a signal with *sigIdx*. The result is stored in *sigName*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigName* is unchanged. The function returns *sigName*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index.

See Also API functions `xPCGetSignalIdx`, `xPCGetSignalWidth`, `xPCGetSignal`, `xPCGetSignals`

Target object properties `ShowSignals`, `Signals`

xPCGetSignals

Purpose Return vector of signal values

Prototype

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

Arguments

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

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

Description The `xPCGetSignals` function is the vector version of the function `xPCGetSignal`. This function returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step (for that, define a scope of type `SCTYPE_HOST` and use `xPCScGetData`). `xPCGetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result. However, the `xPCGetSignals` function is faster, and the signal values are more likely to be spaced closely together. The signals are converted to doubles regardless of the actual data type of the signal.

For *signals*, the list you provide should be stored in an integer array. Get the signal numbers with the function `xPCGetSignalIdx`.

See Also API function `xPCGetSignal`, `xPCGetSignalIdx`

Example To reference signal vector data rather than scalar values, pass a vector of indices for the signal data. For example:


```
/* ***** */

/* Assume a signal of width 10, with the blockpath
 * mySubsys/mySignal and the signal index s1.
 */

int i;
int sigId[10];
double sigVal[10]; /* Signal values are stored here */

/* Get the ID of the first signal */
sigId[0] = xPCGetSignalIdx(port, "mySubsys/mySignal/s1");

if (sigId[0] == -1) {
    /* Handle error */
}

for (i = 1; i < 10; i++) {
    sigId[i] = sigId[0] + i;
}

xPCGetSignals(port, 10, sigId, sigVal);
/* If no error, sigVal should have the signal values */

/* ***** */
```

To repeatedly get the signals, repeat the call to `xPCGetSignals`. If you do not change `sigID`, you only need to call `xPCGetSignalIdx` once.

xPCGetSignalWidth

Purpose Return width of signal

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter the index of a signal.

Return The `xPCGetSignalWidth` function returns the signal width for a signal with *sigIdx*. If the function detects an error, it returns -1.

Description The `xPCGetSignalWidth` function returns the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector again. A signal's width is the number of signals in the vector.

See Also API functions `xPCGetSignalIdx`, `xPCGetSignalName`, `xPCGetSignal`, `xPCGetSignals`

Purpose Copy state log values to array

Prototype

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>state_id</i>	Enter a state identification number.
<i>state_data</i>	The log is stored in <i>state_data</i> , whose allocation is the responsibility of the caller.

Description

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

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

xPCGetStateLog

See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumStates`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Target object method `xpctarget.xpc.getlog`

Target object property `StateLog`

Purpose Return stop time

Prototype `double xPCGetStopTime(int port);`

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

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

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

See Also API function `xPCSetStopTime`
Target object property `StopTime`

xPCGetTargetVersion

Purpose Get xPC Target kernel version

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ver</i>	The version is stored in <i>ver</i> .

Description The `xPCGetTargetVersion` function gets a string with the version number of the xPC Target kernel on the target computer. It then copies that version number into *ver*.

See Also `xPCGetAPIVersion`

Purpose Copy TET log to array

Prototype

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

Arguments

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

Description The `xPCGetTETLog` function gets the task execution time (TET) log. It then copies the log into *TET_data*. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

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

See Also API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTimeLog`

Target object method `xpctarget.xpc.getlog`

Target object property `TETLog`

xPCGetTimeLog

Purpose Copy time log to array

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

Arguments

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

Description The `xPCGetTimeLog` function gets the time log and copies the log into *time_data*. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

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

See Also API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

Target object method `xpctarget.xpc.getlog`

Target object property `TimeLog`

Purpose	Initialize xPC Target DLL
Prototype	<code>int xPCInitAPI(void);</code>
Return	The xPCInitAPI function returns 0 if it completes execution without detecting an error. If the function detects an error, it returns -1.
Description	The xPCInitAPI function initializes the xPC Target dynamic link library. You must execute this function once at the beginning of the application to load the xPC Target API DLL. This function is defined in the file <code>xpcinitfree.c</code> . Link this file with your application.
See Also	API functions <code>xPCFreeAPI</code> , <code>xPCNumLogWraps</code> , <code>xPCNumLogSamples</code> , <code>xPCMaxLogSamples</code> , <code>xPCGetStateLog</code> , <code>xPCGetTETLog</code> , <code>xPCSetLogMode</code> , <code>xPCGetLogMode</code>

xPCIsAppRunning

Purpose	Return target application running status
Prototype	<code>int xPCIsAppRunning(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	If the target application is stopped, the <code>xPCIsAppRunning</code> function returns 0. If the target application is running, this function returns 1. If the function detects an error, it returns -1.
Description	The <code>xPCIsAppRunning</code> function returns 1 or 0 depending on whether the target application is stopped or running. If the function detects is an error, use the function <code>xPCGetLastError</code> to check for the error string constant.
See Also	API function <code>xPCIsOverloaded</code> Target object property <code>Status</code>

Purpose	Return target computer overload status
Prototype	<code>int xPCIsOverloaded(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	If the target application has overloaded the CPU, the <code>xPCIsOverloaded</code> function returns 1. If it has not overloaded the CPU, the function returns 0. If this function detects error, it returns -1.
Description	The <code>xPCIsOverloaded</code> function checks if the target application has overloaded the target computer and returns 1 if it has and 0 if it has not. If the target application is not running, the function returns 0.
See Also	API function <code>xPCIsAppRunning</code> Target object property <code>CPUOverload</code>

xPCIsScFinished

Purpose Return data acquisition status for scope

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

Arguments

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

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

Description The `xPCIsScFinished` function returns a Boolean value depending on whether scope *scNum* is finished (state of `SCST_FINISHED`) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state. Use the `xPCGetScope` function to get the scope number.

See Also API function `xPCScGetState`
Scope object property `Status`

Purpose Load target application onto target computer

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

Arguments

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

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

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

xPCLoadApp

See Also

API functions `xPCStartApp`, `xPCStopApp`, `xPCUnloadApp`,
`xPCSetLoadTimeOut`, `xPCGetLoadTimeOut`

Target object method `xpctarget.xpc.load`

Purpose Restore parameter values

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of the file that contains the saved parameters.

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

See Also API function `xPCSaveParamSet`

xPCMaxLogSamples

Purpose	Return maximum number of samples that can be in log buffer
Prototype	<code>int xPCMaxLogSamples(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCMaxLogSamples</code> function returns the total number of samples. If the function detects an error, it returns -1.
Description	The <code>xPCMaxLogSamples</code> function returns the total number of samples that can be returned in the logging buffers.
See Also	API functions <code>xPCNumLogSamples</code> , <code>xPCNumLogWraps</code> , <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> Target object property <code>MaxLogSamples</code>

Purpose Copy maximum task execution time to array

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	Array of at least two doubles.

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

See Also API functions `xPCMinimumTET`, `xPCAverageTET`
Target object property `MaxTET`

xPCMinimumTET

Purpose Copy minimum task execution time to array

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	Array of at least two doubles.

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

See Also API functions `xPCMaximumTET`, `xPCAverageTET`
Target object property `MinTET`

Purpose	Return number of samples in log buffer
Prototype	<code>int xPCNumLogSamples(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCNumLogSamples</code> function returns the number of samples in the log buffer. If the function detects an error, it returns -1.
Description	<p>The <code>xPCNumLogSamples</code> function returns the number of samples in the log buffer. In contrast to <code>xPCMaxLogSamples</code>, which returns the maximum number of samples that can be logged (because of buffer size constraints), <code>xPCNumLogSamples</code> returns the number of samples actually logged.</p> <p>Note that the target application must be stopped before you get the number.</p>
See Also	API functions <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> , <code>xPCMaxLogSamples</code>

xPCNumLogWraps

Purpose	Return number of times log buffer wraps
Prototype	<code>int xPCNumLogWraps(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCNumLogWraps</code> function returns the number of times the log buffer wraps. If the function detects an error, it returns -1.
Description	The <code>xPCNumLogWraps</code> function returns the number of times the log buffer wraps.
See Also	API functions <code>xPCNumLogSamples</code> , <code>xPCMaxLogSamples</code> , <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> Target object property <code>NumLogWraps</code>

Purpose Open connection to target computer

Prototype `void xPCOpenConnection(int port);`

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

Description The `xPCOpenConnection` function opens a connection to the target computer whose data is indexed by *port*. Before calling this function, set up the target information by calling `xPCRegisterTarget`. A call to either `xPCOpenSerialPort` or `xPCOpenTcpIpPort` can also set up the target information. If the port is already open, calling this function has no effect.

See Also API functions `xPCOpenTcpIpPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCTargetPing`, `xPCCloseConnection`, `xPCRegisterTarget`

xPCOpenSerialPort

Purpose Open RS-232 connection to xPC Target system

Prototype `int xPCOpenSerialPort(int comPort, int baudRate);`

Arguments

<i>comPort</i>	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
<i>baudRate</i>	<i>baudRate</i> must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Return The xPCOpenSerialPort function returns the port value for the connection. If the function detects an error, it returns -1.

Description The xPCOpenSerialPort function initiates an RS-232 connection to an xPC Target system. It returns the port value for the connection. Be sure to pass this value to all the xPC Target API functions that require a port value.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

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

See Also API functions xPCOpenTcpIpPort, xPCClosePort, xPCReOpenPort, xPCTargetPing, xPCOpenConnection, xPCCloseConnection, xPCRegisterTarget, xPCDeRegisterTarget

Purpose	Open TCP/IP connection to xPC Target system				
Prototype	<pre>int xPCOpenTcpIpPort(const char *ipAddress, const char *ipPort);</pre>				
Arguments	<table><tr><td><i>ipAddress</i></td><td>Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".</td></tr><tr><td><i>ipPort</i></td><td>Enter the associated IP port as a string. For example, "22222".</td></tr></table>	<i>ipAddress</i>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".	<i>ipPort</i>	Enter the associated IP port as a string. For example, "22222".
<i>ipAddress</i>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".				
<i>ipPort</i>	Enter the associated IP port as a string. For example, "22222".				
Return	The xPCOpenTcpIpPort function returns a nonnegative integer that you can then use as the port value for an xPC Target API function that requires it. If this operation fails, this function returns -1.				
Description	The xPCOpenTcpIpPort function opens a connection to the TCP/IP location specified by the IP address. It returns a nonnegative integer if it succeeds. Use this integer as the <i>ipPort</i> variable in the xPC Target API functions that require a port value. The global error number is also set, which you can get using xPCGetLastError.				
See Also	API functions xPCOpenSerialPort, xPCClosePort, xPCReOpenPort, xPCTargetPing				

xPCReboot

Purpose Reboot target computer

Prototype `void xPCReboot(int port);`

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

Description The `xPCReboot` function reboots the target computer. This function returns nothing. This function does not close the connection to the target computer. You should either explicitly close the port or call `xPCReOpenPort` once the target computer has rebooted.

See Also API function `xPCReOpenPort`
Target object method `xpctarget.xpc.reboot`

Purpose	Reopen communication channel
Prototype	<code>int xPCReOpenPort(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCReOpenPort</code> function returns 0 if it reopens a connection without detecting an error. If the function detects an error, it returns -1.
Description	The <code>xPCReOpenPort</code> function reopens the communications channel pointed to by <i>port</i> . The difference between this function and <code>xPCOpenSerialPort</code> or <code>xPCOpenTcpIpPort</code> is that <code>xPCReOpenPort</code> uses the already existing settings, while the other functions need to set up the port.
See Also	API functions <code>xPCOpenTcpIpPort</code> , <code>xPCClosePort</code>

xPCRegisterTarget

Purpose Register target with xPC Target API library

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

Arguments *commType* Specify the communication type (TCP/IP or RS-232) between the host and the target.

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

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

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

comPort *comPort* and *baudRate* are as in xPCOpenSerialPort.

baudRate The *baudRate* must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Return The xPCRegisterTarget function returns the port number. If the function detects an error, it returns -1.

Description The xPCRegisterTarget function works similarly to xPCOpenSerialPort and xPCOpenTcpIpPort, except that it does not try to open a connection to the target computer. In other words, xPCOpenSerialPort or xPCOpenTcpIpPort is equivalent to calling xPCRegisterTarget with the required parameters, followed by a call to xPCOpenConnection.

Use the constants COMMTYP_TCP/IP and COMMTYP_RS232 for *commType*. If *commType* is set to COMMTYP_RS232, the function ignores *ipAddress*

and *ipPort*. Analogously, the function ignores *comPort* and *baudRate* if *commType* is set to `COMMTYP_TCPIP`.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

See Also

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

xPCRemScope

Purpose Remove scope

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

Arguments

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

Description The `xPCRemScope` function removes the scope with number *scNum*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, see `xPCGetScopes`. Use the `xPCGetScope` function to get the scope number.

See Also API functions `xPCAddScope`, `xPCScRemSignal`, `xPCGetScopes`
Target object method `xpctarget.xpc.remscope`

Purpose Save parameter values of target application

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

Arguments

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

filename Enter the name of the file to contain the saved parameters.

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

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

See Also API function `xPCLoadParamSet`

xPCScAddSignal

Purpose Add signal to scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

Description The `xPCScAddSignal` function adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScGetSignals` to get a list of the signals already present. Use the function `xPCGetScope` to get the scope number. Use the `xPCGetSignalIdx` function to get the signal number.

See Also API functions `xPCScRemSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`
Scope object method `xpctarget.xpcsc.addsignal`

Purpose Scope autorestart status

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

Arguments

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

Return The `xPCScGetAutoRestart` function returns the autorestart flag value of scope *scNum*. If the function detects an error, it returns -1.

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

See Also API functions `xPCScSetAutoRestart`

xPCScGetData

Purpose Copy scope data to array

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

Arguments

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

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

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

See Also

API functions `xPCGetScope`, `xPCScGetState`, `xPCScGetSignals`

Scope object property `Data`

xPCScGetDecimation

Purpose Return decimation of scope

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

Arguments

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

Return The `xPCScGetDecimation` function returns the decimation of scope *scNum*. If the function detects an error, it returns -1.

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

See Also API function `xPCScSetDecimation`
Scope object property `Decimation`

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

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

Arguments

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

Return The `xPCScGetNumPrePostSamples` function returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this function returns the minimum integer value (-2147483647-1).

Description The `xPCScGetNumPrePostSamples` function gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples. Use the `xPCGetScope` function to get the scope number.

See Also API function `xPCScSetNumPrePostSamples`
Scope object property `NumPrePostSamples`

xPCScGetNumSamples

Purpose Get number of samples in one data acquisition cycle

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

Arguments

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

Return The `xPCScGetNumSamples` function returns the number of samples in the scope *scNum*. If the function detects an error, it returns -1.

Description The `xPCScGetNumSamples` function gets the number of samples in one data acquisition cycle for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also API function `xPCScSetNumSamples`
Scope object property `NumSamples`

Purpose	Get number of signals in scope				
Prototype	<code>int xPCScGetNumSignals(int <i>port</i>, int <i>scNum</i>);</code>				
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>scNum</i></td><td>Enter the scope number.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>scNum</i>	Enter the scope number.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>scNum</i>	Enter the scope number.				
Return	The <code>xPCScGetNumSignals</code> function returns the number of signals in the scope <i>scNum</i> . If the function detects an error, it returns -1.				
Description	The <code>xPCScGetNumSignals</code> function gets the number of signals in the scope <i>scNum</i> . Use the <code>xPCGetScope</code> function to get the scope number.				
See Also	API function <code>xPCGetScope</code>				

xPCScGetSignalList

Purpose Copy list of signals to array

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

Arguments

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

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

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

Purpose Copy list of signals to array

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

Arguments

<i>port</i>	Value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	Integer array allocated by the caller as a list containing the signal identifiers, terminated by -1.

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

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

See Also API functions `xPCScGetData`, `xPCGetScopes`
Scope object property `Signals`

xPCScGetStartTime

Purpose Get start time for last data acquisition cycle

Prototype `double xPCScGetStartTime(int port, int scNum);`

Arguments

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

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

Description The `xPCScGetStartTime` function gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`. Use the `xPCGetScope` function to get the scope number.

See Also API functions `xPCScGetNumSamples`, `xPCScGetDecimation`

Purpose Get state of scope

Prototype `int xPCScGetState(int port, int scNum);`

Arguments

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

scNum Enter the scope number.

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

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

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

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

xPCScGetState

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

See Also

API functions `xPCScStart`, `xPCScStop`

Scope object property `Status`

Purpose Get trigger level for scope

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

Arguments

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

Return The `xPCScGetTriggerLevel` function returns the scope trigger level. If the function detects an error, it returns -1.

Description The `xPCScGetTriggerLevel` function gets the trigger level for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also API functions `xPCScSetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`
Scope object property `TriggerLevel`

xPCScGetTriggerMode

Purpose Get trigger mode for scope

Prototype `int xPCScGetTriggerMode(int port, int scNum);`

Arguments

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

Return The `xPCScGetTriggerMode` function returns the scope trigger mode. If the function detects an error, it returns -1.

Description The `xPCScGetTriggerMode` function gets the trigger mode for scope *scNum*. Use the `xPCGetScope` function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.

Constant	Value	Description
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`

Scope object method `trigger`

Scope object property `TriggerMode`

xPCScGetTriggerScope

Purpose	Get trigger scope				
Prototype	<code>int xPCScGetTriggerScope(int <i>port</i>, int <i>scNum</i>);</code>				
Arguments	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>scNum</i></td><td>Enter the scope number.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>scNum</i>	Enter the scope number.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>scNum</i>	Enter the scope number.				
Return	The <code>xPCScGetTriggerScope</code> function returns a trigger scope. If the function detects an error, it returns -1.				
Description	The <code>xPCScGetTriggerScope</code> function gets the trigger scope for scope <i>scNum</i> . Use the <code>xPCGetScope</code> function to get the scope number.				
See Also	API functions <code>xPCScSetTriggerLevel</code> , <code>xPCScGetTriggerLevel</code> , <code>xPCScSetTriggerSlope</code> , <code>xPCScGetTriggerSlope</code> , <code>xPCScSetTriggerSignal</code> , <code>xPCScGetTriggerSignal</code> , <code>xPCScSetTriggerMode</code> , <code>xPCScGetTriggerMode</code> Scope object property <code>TriggerScope</code>				

Purpose Get sample number for triggering scope

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

Arguments

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

Return The `xPCScGetTriggerScopeSample` function returns a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the function detects an error, it returns `INT_MIN` (-2147483647-1).

Description The `xPCScGetTriggerScopeSample` function gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScSetTriggerScopeSample`

Scope object property `TriggerSample`

xPCScGetTriggerSignal

Purpose Get trigger signal for scope

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

Arguments

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

Return The `xPCScGetTriggerSignal` function returns the scope trigger signal. If the function detects an error, it returns -1.

Description The `xPCScGetTriggerSignal` function gets the trigger signal for scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`
Scope object method `trigger`
Scope object property `TriggerSignal`

Purpose Get trigger slope for scope

Prototype `int xPCScGetTriggerSlope(int port, int scNum);`

Arguments

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

Return The `xPCScGetTriggerSlope` function returns the scope trigger slope. If the function detects an error, it returns -1.

Description The `xPCScGetTriggerSlope` function gets the trigger slope of scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope. Use the constants defined in `xpcapiconst.h` to interpret the trigger slope. These constants have the following meanings:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger slope must be rising when the signal crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger slope must be falling when the signal crosses the trigger value.

xPCScGetTriggerSlope

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`,
`xPCScSetTriggerSlope`, `xPCScSetTriggerSignal`,
`xPCScGetTriggerSignal`, `xPCScSetTriggerScope`,
`xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object method `xpctarget.xpcsc.trigger`

Scope object properties `TriggerMode`, `TriggerSlope`

Purpose Get type of scope

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

Arguments

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

Return The `xPCScGetType` function returns the scope type. If the function detects an error, it returns -1.

Description The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the host computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

See Also API functions `xPCAddScope`, `xPCRemScope`
Scope object property `Type`

xPCScRemSignal

Purpose Remove signal from scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

Description The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCGetScopes` to determine the existing scopes, and use `xPCScGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also API functions `xPCScAddSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`, `xPCScGetSignals`, `xPCScGetState`
Scope object method `remsignal`

Purpose Scope autorestart status

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>autorestart</i>	Enter value to enable (1) or disable (0) scope autorestart.

Description The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to 0 or 1. 0 disables the flag, 1 enables it. Use this function only when the scope is stopped.

See Also API functions `xPCScGetAutoRestart`

xPCScSetDecimation

Purpose Set decimation of scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>decimation</i>	Enter an integer for the decimation.

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

See Also API functions `xPCScGetDecimation`, `xPCScGetState`
Scope object property `Decimation`

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

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>prepost</i>	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

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

See Also API functions `xPCScGetNumPrePostSamples`, `xPCScGetState`
Scope object property `NumPrePostSamples`

xPCScSetNumSamples

Purpose Set number of samples in one data acquisition cycle

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>samples</i>	Enter the number of samples you want to acquire in one cycle.

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

See Also API functions `xPCScGetNumSamples`, `xPCScGetState`
Scope object property `NumSamples`

Purpose

Set trigger level for scope

Prototype

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

Arguments

<i>port</i>	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>level</i>	Value for a signal to trigger data acquisition with a scope.

Description

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

See Also

API functions xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScGetState

Scope object property TriggerLevel

xPCScSetTriggerMode

Purpose Set trigger mode of scope

Prototype `void xPCScSetTriggerMode(int port, int scNum, int mode);`

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Trigger mode for a scope.

Description The `xPCScSetTriggerMode` function sets the trigger mode of scope *scNum* to *mode*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScGetTriggerMode, xPCScGetState

Scope object method trigger

Scope object property TriggerMode

xPCScSetTriggerScope

Purpose Select scope to trigger another scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigScope</i>	Enter the scope number of the scope used for a trigger.

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

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

See Also API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Scope object property `TriggerScope`

Purpose	Set sample number for triggering scope
Prototype	<pre>void xPCScSetTriggerScopeSample(int port, int scNum, int trigScSamp);</pre>
Arguments	<p><i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</p> <p><i>scNum</i> Enter the scope number.</p> <p><i>trigScSamp</i> Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.</p>
Description	<p>The <code>xPCScSetTriggerScopeSample</code> function sets the number of samples (<i>trigScSamp</i>) a triggering scope acquires before it triggers a second scope (<i>scNum</i>). Use the <code>xPCGetScopes</code> function to get a list of scopes.</p> <p>For meaningful results, set <i>trigScSamp</i> between -1 and (<i>nSamp</i>-1). <i>nSamp</i> 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.</p> <p>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 <i>trigScSamp</i>.</p>
See Also	<p>API functions <code>xPCGetScopes</code>, <code>xPCScSetTriggerLevel</code>, <code>xPCScGetTriggerLevel</code>, <code>xPCScSetTriggerSlope</code>, <code>xPCScGetTriggerSlope</code>, <code>xPCScSetTriggerSignal</code>, <code>xPCScGetTriggerSignal</code>, <code>xPCScSetTriggerScope</code>, <code>xPCScGetTriggerScope</code>, <code>xPCScSetTriggerMode</code>, <code>xPCScGetTriggerMode</code>, <code>xPCScGetTriggerScopeSample</code></p> <p>Scope object properties <code>TriggerMode</code>, <code>TriggerSample</code></p>

xPCScSetTriggerSignal

Purpose Select signal to trigger scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSig</i>	Enter a signal number.

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

See Also API functions `xPCGetScopes`, `xPCScGetState`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`
Scope object property `TriggerSignal`

Purpose Set slope of signal that triggers scope

Prototype `void xPCScSetTriggerSlope(int port, int scNum, int trigSlope);`

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSlope</i>	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
<code>TRIGSLOPE_EITHER</code>	0	The trigger slope can be either rising or falling.
<code>TRIGSLOPE_RISING</code>	1	The trigger signal value must be rising when it crosses the trigger value.
<code>TRIGSLOPE_FALLING</code>	2	The trigger signal value must be falling when it crosses the trigger value.

xPCScSetTriggerSlope

See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScGetState

Scope object property TriggerSlope

Purpose Set software trigger of scope

Prototype `void xPCScSoftwareTrigger(int port, int scNum);`

Arguments

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

You can use the `xPCScSoftwareTrigger` function to trigger the scope, regardless of the trigger mode.

See Also API functions `xPCGetScopes`, `xPCScGetState`, `xPCIsScFinished`
Scope object method `trigger`
Scope object property `TriggerMode`

xPCScStart

Purpose Start data acquisition for scope

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

Arguments

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

Description The `xPCScStart` function starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call `xPCScGetState` to check the state of the scope or, for host scopes that are already started, call `xPCIsScFinished`. Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCScGetState`, `xPCScStop`, `xPCIsScFinished`
Scope object method `start` (scope object)

Purpose Stop data acquisition for scope

Prototype `void xPCScStop(int port, int scNum);`

Arguments

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

scNum Enter the scope number.

Description The `xPCScStop` function stops the scope *scNum*. This sets the scope to the "Interrupted" state. The scope must be running for this function to succeed. Use `xPCScGetState` to determine the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCScStart`, `xPCScGetState`
Scope object method `stop` (scope object)

xPCSetEcho

Purpose Turn message display on or off

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>mode</i>	Valid values are
0	Turns the display off
1	Turns the display on

Description On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the target application is stopped. When you turn the message display off, the message screen no longer updates.

See Also API function `xPCGetEcho`

Purpose Set last error to specific string constant

Prototype `void xPCSetLastError(int error);`

Arguments *error* Specify the string constant for the error.

Description The xPCSetLastError function sets the global error constant returned by xPCGetLastError to *error*. This is useful only to set the string constant to ENOERR, indicating no error was found.

See Also API functions xPCGetLastError, xPCErrorMsg

xPCSetLoadTimeOut

Purpose Change initialization timeout value between host computer and target computer

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>timeOut</i>	Enter the new communication timeout value.

Description The `xPCSetLoadTimeOut` function changes the timeout value for communication between the host computer and target computer. The *timeOut* value is the time an xPC Target API function waits for the communication between the host computer and target computer to complete before returning. It enables you to set the number of communication attempts to be made before signaling a timeout.

For example, the function `xPCLoadApp` waits to check whether the model initialization for a new application is complete before returning. When a new target application is loaded onto the target computer, the function `xPCLoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCLoadApp` returns a timeout error.

By default, `xPCLoadApp` checks for target readiness for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, models with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. Other functions that communicate with the target computer will wait for *timeOut* seconds before declaring a timeout event.

See Also API functions `xPCGetLoadTimeOut`, `xPCLoadApp`, `xPCUnloadApp`

Purpose Set logging mode and increment value of scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>logging_data</i>	Logging mode and increment value.

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

See Also

- API function `xPCGetLogMode`
- API structure `lgmode`
- Target object property `LogMode`

xPCSetParam

Purpose Change value of parameter

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Parameter index.
<i>paramValue</i>	Vector of doubles, assumed to be the size required by the parameter type

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

See Also API functions `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetParam`

Purpose Change target application sample time

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ts</i>	Sample time for the target application.

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

See Also API function `xPCGetSampleTime`
Target object property `SampleTime`

xPCSetScope

Purpose Set properties of scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>state</i>	Enter a structure of type <code>scopedata</code> .

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

See Also API functions `xPCGetScope`, `xPCScGetState`, `scopedata`
Scope object method `set (scope object)`

Purpose Change target application stop time

Prototype `void xPCSetStopTime(int port, double tfinal);`

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>tfinal</i>	Enter the stop time, in seconds.

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

See Also API function `xPCGetStopTime`
Target object property `StopTime`

xPCStartApp

Purpose Start target application

Prototype `void xPCStartApp(int port);`

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

Description The `xPCStartApp` function starts the target application loaded on the target machine.

See Also API function `xPCStopApp`
Target object method `start` (target application object)

Purpose Stop target application

Prototype void xPCStopApp(int *port*);

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

Description The xPCStopApp function stops the target application loaded on the target computer. The target application remains loaded and the parameter changes you made remain intact. If you want to stop and unload an application, use xPCUnloadApp.

See Also API functions xPCStartApp, xPCUnloadApp
Target object method stop (target application object)

xPCTargetPing

Purpose	Ping target computer
Prototype	<code>int xPCTargetPing(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCTargetPing</code> function does not return an error status. This function returns 1 if the target responds. If the target computer does not respond, the function returns 0.
Description	<p>The <code>xPCTargetPing</code> function pings the target computer and returns 1 or 0 depending on whether the target responds or not. This function returns an error string constant only when there is an error in the input parameter (for example, the port number is invalid or <i>port</i> is not open). Other errors, such as the inability to connect to the target, are ignored.</p> <p>If you are using TCP/IP, note that <code>xPCTargetPing</code> will cause the target computer to close the TCP/IP connection. You can use <code>xPCOpenConnection</code> to reconnect. You can also use this <code>xPCTargetPing</code> feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if your host side program crashes).</p>
See Also	API functions <code>xPCOpenConnection</code> , <code>xPCOpenSerialPort</code> , <code>xPCOpenTcpIpPort</code> , <code>xPCClosePort</code>

Purpose Get status of grid line for particular scope

Prototype `int xPCTgScGetGrid(int port, int scNum);`

Arguments

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

Return Returns the status of the grid for a scope of type `SCTYPE_TARGET`. If the function detects an error, it returns -1.

Description The `xPCTgScGetGrid` function gets the state of the grid lines for scope *scNum* (which must be of type `SCTYPE_TARGET`). A return value of 1 implies grid on, while 0 implies grid off. Note that when the scope mode is set to `SCMODE_NUMERICAL`, the grid is not drawn even when the grid mode is set to 1.

Tip

- Use `xPCTgScSetMode` and `xPCTgScGetMode` to set and retrieve the scope mode.
 - Use `xPCGetScopes` to get a list of scopes.
-

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

xPCTgScGetMode

Purpose Get scope mode for displaying signals

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

Arguments

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

Return The `xPCTgScGetMode` function returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

If this function detects an error, it returns -1.

Description The `xPCTgScGetMode` function gets the mode (`SCMODE_NUMERICAL`, `SCMODE_REDRAW`, `SCMODE_SLIDING`, `SCMODE_ROLLING`) of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`

Purpose	Get view mode for target computer display
Prototype	<code>int xPCTgScGetViewMode(int <i>port</i>);</code>
Arguments	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
Return	The <code>xPCTgScGetViewMode</code> function returns the view mode for the target computer screen. If the function detects an error, it returns -1.
Description	The <code>xPCTgScGetViewMode</code> function gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is that of the scope currently displayed on the screen. If the value is 0, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).
See Also	API functions <code>xPCGetScopes</code> , <code>xPCTgScSetGrid</code> , <code>xPCTgScGetGrid</code> , <code>xPCTgScSetViewMode</code> , <code>xPCTgScSetMode</code> , <code>xPCTgScGetMode</code> , <code>xPCTgScSetYLimits</code> , <code>xPCTgScGetYLimits</code> Target object property <code>ViewMode</code>

xPCTgScGetYLimits

Purpose Copy *y*-axis limits for scope to array

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>limits</i>	The first element of the array is the lower limit while the second element is the upper limit.

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

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`

Scope object property `YLimit`

Purpose Set grid mode for scope

Prototype void xPCTgScSetGrid(int *port*, int *scNum*, int *grid*);

Arguments

<i>port</i>	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>grid</i>	Enter a grid value.

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

See Also API functions xPCGetScopes, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScSetMode, xPCTgScGetMode, xPCTgScSetYLimits, xPCTgScGetYLimits

Scope object property Grid

xPCTgScSetMode

Purpose Set display mode for scope

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

Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Enter the value for the mode.

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

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`

Purpose Set view mode for scope

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

Arguments

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

Description The `xPCTgScSetViewMode` function sets the target computer screen to display one scope with scope number *scNum*. If you set *scNum* to 0, the target computer screen displays all the defined scopes. Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`
Target object property `ViewMode`

xPCTgScSetYLimits

Purpose Set *y*-axis limits for scope

Prototype `void xPCTgScSetYLimits(int port, int scNum, const double *Ylimits);`

Arguments

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

scNum Enter the scope number.

Ylimits Enter a two-element array.

Description The `xPCTgScSetYLimits` function sets the *y*-axis limits for a scope with scope number *scNum* and type `SCTYPE_TARGET` to the values in the double array *Ylimits*. The first element is the lower limit, and the second element is the upper limit. Set both limits to 0.0 to specify autoscaling. Use the `xPCGetScopes` function to get a list of scopes.

See Also API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScGetYLimits`

Scope object property `Ylimit`

Purpose Unload target application

Prototype `void xPCUnloadApp(int port);`

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

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

See Also API function `xPCLoadApp`
Target object methods `xpctarget.xpc.load`, `xpctarget.xpc.unload`

xPCUnloadApp

xPC Target API Reference for COM

- “COM API Methods” on page 8-2
- “COM API Methods — Alphabetical List” on page 8-9

COM API Methods

In this section...

“Target Computers” on page 8-2
 “Target Applications” on page 8-3
 “Scopes” on page 8-4
 “Parameters” on page 8-6
 “Signals” on page 8-6
 “Data Logs” on page 8-7
 “File Systems” on page 8-7
 “Errors” on page 8-8

Note The xPC Target COM API is no longer being enhanced. You should use the xPC Target API for Microsoft .NET Framework instead. See “xPC Target API for Microsoft .NET Framework” on page 1-3

Target Computers

xPCProtocol.Close	Close RS-232 or TCP/IP communication connection
xPCProtocol.GetLoadTimeOut	Return current timeout value for target application initialization
xPCProtocol.Init	Initialize xPC Target API DLL
xPCProtocol.Port	Contain communication channel index
xPCProtocol.Reboot	Reboot target computer
xPCProtocol.RS232Connect	Open RS-232 connection to target computer
xPCProtocol.SetLoadTimeOut	Change initialization timeout value
xPCProtocol.TargetPing	Ping target computer

xPCProtocol.TcpIpConnect	Open TCP/IP connection to target computer
xPCProtocol.Term	Unload xPC Target API DLL from memory
xPCTarget.UnLoadApp	Unload target application

Target Applications

xPCTarget.AverageTET	Get average task execution time
xPCTarget.GetAppName	Get target application name
xPCTarget.GetExecTime	Get execution time for target application
xPCTarget.GetNumOutputs	Get number of outputs
xPCTarget.GetSampleTime	Get sample time
xPCTarget.GetStopTime	Get stop time
xPCTarget.Init	Initialize target object to communicate with target computer
xPCTarget.IsAppRunning	Return running status for target application
xPCTarget.IsOverloaded	Return overload status for target computer
xPCTarget.MaximumTET	Copy maximum task execution time to array
xPCTarget.MaxLogSamples	Return maximum number of samples that can be in log buffer
xPCTarget.MinimumTET	Copy minimum task execution time to array
xPCTarget.NumLogSamples	Return number of samples in log buffer
xPCTarget.NumLogWraps	Return number of times log buffer wraps

xPCTarget.SetSampleTime	Change sample time for target application
xPCTarget.SetStopTime	Change stop time of target application
xPCTarget.StartApp	Start target application
xPCTarget.StopApp	Stop target application

Scopes

xPCScopes.AddFileScope	Create new file scope
xPCScopes.AddHostScope	Create new host scope
xPCScopes.AddTargetScope	Create new target scope
xPCScopes.GetScopes	Get and copy list of scope numbers
xPCScopes.Init	Initialize scope object to communicate with target computer
xPCScopes.IsScopeFinished	Get data acquisition status for scope
xPCScopes.RemScope	Remove scope
xPCScopes.ScopeAddSignal	Add signal to scope
xPCScopes.ScopeGetAutoRestart	Scope autorestart value
xPCScopes.ScopeGetData	Copy scope data to array
xPCScopes.ScopeGetDecimation	Get decimation of scope
xPCScopes.ScopeGetNumPrePostSamples	Get number of pre- or posttriggering samples before triggering scope
xPCScopes.ScopeGetNumSamples	Get number of samples in one data acquisition cycle
xPCScopes.ScopeGetSignals	Get list of signals
xPCScopes.ScopeGetStartTime	Get last data acquisition cycle start time
xPCScopes.ScopeGetState	Get state of scope
xPCScopes.ScopeGetTriggerLevel	Get trigger level for scope

xPCScopes.ScopeGetTriggerMode	Get trigger mode for scope
xPCScopes.ScopeGetTriggerModeStr	Get trigger mode as string
xPCScopes.ScopeGetTriggerSample	Get sample number for triggering scope
xPCScopes.ScopeGetTriggerSignal	Get trigger signal for scope
xPCScopes.ScopeGetTriggerSlope	Get trigger slope for scope
xPCScopes.ScopeGetTriggerSlopeStr	Get trigger slope as string
xPCScopes.ScopeGetType	Get type of scope
xPCScopes.ScopeRemSignal	Remove signal from scope
xPCScopes.ScopeSetAutoRestart	Scope autorestart value
xPCScopes.ScopeSetDecimation	Set decimation of scope
xPCScopes.ScopeSetNumPrePostSamples	Set number of pre- or posttriggering samples before triggering scope
xPCScopes.ScopeSetNumSamples	Set number of samples in one data acquisition cycle
xPCScopes.ScopeSetTriggerLevel	Set trigger level for scope
xPCScopes.ScopeSetTriggerMode	Set trigger mode of scope
xPCScopes.ScopeSetTriggerSample	Set sample number for triggering scope
xPCScopes.ScopeSetTriggerSignal	Select signal to trigger scope
xPCScopes.ScopeSetTriggerSlope	Set slope of signal that triggers scope
xPCScopes.ScopeSoftwareTrigger	Set software trigger of scope
xPCScopes.ScopeStart	Start data acquisition for scope
xPCScopes.ScopeStop	Stop data acquisition for scope
xPCScopes.TargetScopeGetGrid	Get status of grid line for particular scope
xPCScopes.TargetScopeGetMode	Get scope mode for displaying signals

xPCScopes.TargetScopeGetModeStr	Get scope mode string for displaying signals
xPCScopes.TargetScopeGetViewMode	Get view mode for target computer display
xPCScopes.TargetScopeGetYLimits	Get y-axis limits for scope
xPCScopes.TargetScopeSetGrid	Set grid mode for scope
xPCScopes.TargetScopeSetMode	Set display mode for scope
xPCScopes.TargetScopeSetViewMode	Set view mode for scope
xPCScopes.TargetScopeSetYLimits	Set y-axis limits for scope

Parameters

xPCTarget.GetNumParams	Get number of tunable parameters
xPCTarget.GetParam	Get parameter values
xPCTarget.GetParamDims	Get row and column dimensions of parameter
xPCTarget.GetParamIdx	Get parameter index
xPCTarget.GetParamName	Get parameter name
xPCTarget.SetParam	Change parameter value

Signals

xPCTarget.GetNumSignals	Get number of signals
xPCTarget.GetSignal	Get signal value
xPCTarget.GetSignalidsfromLabel	Get signal IDs from signal label
xPCTarget.GetSignalIdx	Get signal index
xPCTarget.GetSignalLabel	Get signal label
xPCTarget.GetSignalName	Copy signal name to character array

xPCTarget.GetSignals	Get vector of signal values
xPCTarget.GetSignalWidth	Get width of signal

Data Logs

xPCTarget.GetNumStates	Get number of states
xPCTarget.GetOutputLog	Copy output log data to array
xPCTarget.GetStateLog	Get state log
xPCTarget.GetTETLog	Get TET log
xPCTarget.GetTimeLog	Get time log

File Systems

FSDir	Type definition for file system folder information structure
FSDiskInfo	Type definition for file system disk information structure
xPCFileSystem.CD	Change current folder on target computer to specified path
xPCFileSystem.CloseFile	Close file on target computer
xPCFileSystem.DirList	Return contents of target computer folder
xPCFileSystem.GetDiskInfo	Return disk information
xPCFileSystem.GetFileSize	Return size of file on target computer
xPCFileSystem.Init	Initialize file system object to communicate with target computer
xPCFileSystem.MKDIR	Create folder on target computer
xPCFileSystem.OpenFile	Open file on target computer
xPCFileSystem.PWD	Get current folder of target computer
xPCFileSystem.ReadFile	Read open file on target computer

xPCFileSystem.RemoveFile	Remove file from target computer
xPCFileSystem.RMDIR	Remove folder from target computer
xPCFileSystem.ScGetFileName	Get name of file for scope
xPCFileSystem.ScGetWriteMode	Get write mode of file for scope
xPCFileSystem.ScGetWriteSize	Get block write size of data chunks
xPCFileSystem.ScSetFileName	Specify file name to contain signal data
xPCFileSystem.ScSetWriteMode	Specify when file allocation table entry is updated
xPCFileSystem.ScSetWriteSize	Specify that memory buffer collect data in multiples of write size
xPCFileSystem.WriteFile	Write to file on target computer

Errors

xPCProtocol.GetxPCErrorMsg	Return error string
xPCProtocol.isxPCError	Return error status
xPCScopes.GetxPCError	Get error string
xPCScopes.isxPCError	Get error status
xPCTarget.GetxPCError	Get error string
xPCTarget.isxPCError	Return error status

COM API Methods – Alphabetical List

FSDir

Purpose Type definition for file system folder information structure

Syntax

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

Fields

<i>Name</i>	This value contains the name of the file or folder.
<i>Date</i>	This value contains the date the file or folder was last modified.
<i>Time</i>	This value contains the time the file or folder was last modified.
<i>Bytes</i>	This value contains the size of the file in bytes. If the element is a folder, this value is 0.
<i>isdir</i>	This value indicates if the element is a file (0) or folder (1). If it is a folder, <i>Bytes</i> has a value of 0.

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

See Also API method `xPCFileSystem.DirList`

Purpose Type definition for file system disk information structure

Syntax

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

Fields

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

<i>FATType</i>	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of <i>Files</i> .

<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to <i>FreeClusters</i> .

Description The FSDiskInfo structure contains information for file system disks.

See Also API method `xPCFileSystem.GetDiskInfo`

xPCFileSystem.CD

Purpose	Change current folder on target computer to specified path
Prototype	<code>long CD(BSTR <i>dir</i>);</code>
Member Of	XPCAPICOMLib.xPCFileSystem
Arguments	[in] <i>dir</i> Enter the path on the target computer to change to.
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.
Description	The xPCFileSystem.CD method changes the current folder on the target computer to the path specified in <i>dir</i> . Use the xPCFileSystem.PWD method to show the current folder of the target computer.
See Also	API method xPCFileSystem.PWD

Purpose	Close file on target computer		
Prototype	<code>CloseFile(long <i>filehandle</i>);</code>		
Member Of	XPCAPICOMLib.xPCFileSystem		
Arguments	<table><tr><td>[in] <i>filehandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	[in] <i>filehandle</i>	Enter the file handle of an open file on the target computer.
[in] <i>filehandle</i>	Enter the file handle of an open file on the target computer.		
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.		
Description	The <code>xPCFileSystem.CloseFile</code> method closes the file associated with <i>fileHandle</i> on the target computer. <i>fileHandle</i> is the handle of a file previously opened by the <code>xPCFileSystem.OpenFile</code> method.		
See Also	API methods <code>xPCFileSystem.OpenFile</code> , <code>xPCFileSystem.ReadFile</code> , <code>xPCFileSystem.WriteFile</code>		

xPCFileSystem.DirList

Purpose Return contents of target computer folder

Prototype `DirList(BSTR path);`

Member Of XPCAPICOMLib.xPCFileSystem

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

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

See Also API structure `FSDir`
API method `xPCFileSystem.GetDiskInfo`

Purpose	Return disk information		
Prototype	<code>GetDiskInfo(BSTR <i>driveLetter</i>);</code>		
Member Of	XPCAPICOMLib.xPCFileSystem		
Arguments	<table><tr><td><code>[in] <i>driveLetter</i></code></td><td>Enter the driver letter that contains the file system.</td></tr></table>	<code>[in] <i>driveLetter</i></code>	Enter the driver letter that contains the file system.
<code>[in] <i>driveLetter</i></code>	Enter the driver letter that contains the file system.		
Description	The <code>xPCFileSystem.GetDiskInfo</code> method accepts as input the drive specified by <i>driveLetter</i> and fills in the fields of the <code>FSDiskInfo</code> structure.		
See Also	API structure <code>FSDiskInfo</code> API method <code>xPCFileSystem.DirList</code>		

xPCFileSystem.GetFileSize

Purpose	Return size of file on target computer		
Prototype	<code>long GetFileSize(long <i>filehandle</i>);</code>		
Member Of	XPCAPICOMLib.xPCFileSystem		
Arguments	<table><tr><td><code>[in] <i>filehandle</i></code></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	<code>[in] <i>filehandle</i></code>	Enter the file handle of an open file on the target computer.
<code>[in] <i>filehandle</i></code>	Enter the file handle of an open file on the target computer.		
Return	This method returns the size of the specified file in bytes.		
Description	The <code>xPCFileSystem.GetFileSize</code> method returns the size, in bytes, of the file associated with <i>filehandle</i> on the target computer. <i>filehandle</i> is the handle of a file previously opened by the <code>xPCFileSystem.OpenFile</code> method.		
See Also	API methods <code>xPCFileSystem.OpenFile</code> , <code>xPCFileSystem.ReadFile</code>		

Purpose	Initialize file system object to communicate with target computer		
Prototype	<code>long Init(IxPCProtocol* xPCProtocol);</code>		
Member Of	XPCAPICOMLib.xPCFileSystem		
Arguments	<table><tr><td><code>[in] xPCProtocol</code></td><td>Specify the communication port of the target computer object for which the file system is to be initialized.</td></tr></table>	<code>[in] xPCProtocol</code>	Specify the communication port of the target computer object for which the file system is to be initialized.
<code>[in] xPCProtocol</code>	Specify the communication port of the target computer object for which the file system is to be initialized.		
Return	If the method detects an error, it returns -1. Otherwise, the xPCFileSystem.Init method returns 0.		
Description	The xPCFileSystem.Init method initializes the file system object to communicate with the target computer referenced by the xPCProtocol object.		

xPCFileSystem.MKDIR

Purpose Create folder on target computer

Prototype long MKDIR(BSTR *dirname*);

Member Of XPCAPICOMLib.xPCFileSystem

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

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

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

See Also API method xPCFileSystem.PWD

Purpose Open file on target computer

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

Member Of XPCAPICOMLib.xPCFileSystem

Arguments

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

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

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

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

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

xPCFileSystem.PWD

Purpose	Get current folder of target computer
Prototype	BSTR PWD();
Member Of	XPCAPICOMLib.xPCFileSystem
Return	This method returns the path of the current folder on the target computer.
Description	The xPCFileSystem.PWD method places the path of the current folder on the target computer.
See Also	API method xPCFileSystem.CD

Purpose	Read open file on target computer						
Prototype	VARIANT ReadFile(int <i>fileHandle</i> , int <i>start</i> , int <i>numbytes</i>);						
Member Of	XPCAPICOMLib.xPCFileSystem						
Arguments	<table><tr><td>[in] <i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr><tr><td>[in] <i>start</i></td><td>Enter an offset from the beginning of the file from which this method can start to read.</td></tr><tr><td>[in] <i>numbytes</i></td><td>Enter the number of bytes this method is to read from the file.</td></tr></table>	[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.	[in] <i>start</i>	Enter an offset from the beginning of the file from which this method can start to read.	[in] <i>numbytes</i>	Enter the number of bytes this method is to read from the file.
[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.						
[in] <i>start</i>	Enter an offset from the beginning of the file from which this method can start to read.						
[in] <i>numbytes</i>	Enter the number of bytes this method is to read from the file.						
Return	This method returns the results of the read operation as a VARIANT of type Byte. If the method detects an error, it returns VT_ERROR, whose value is 10, instead.						
Description	The xPCFileSystem.ReadFile method reads an open file on the target computer and returns the results of the read operation as a VARIANT of type Byte. <i>fileHandle</i> is the file handle of a file previously opened by xPCFileSystem.OpenFile. You can specify that the read operation begin at the beginning of the file (default) or at a certain offset into the file (<i>start</i>). The <i>numbytes</i> parameter specifies how many bytes the xPCFileSystem.ReadFile method is to read from the file.						
See Also	API methods xPCFileSystem.CloseFile, xPCFileSystem.GetFileSize, xPCFileSystem.OpenFile, xPCFileSystem.WriteFile						

xPCFileSystem.RemoveFile

Purpose Remove file from target computer

Prototype long RemoveFile(BSTR *filename*);

Member Of XPCAPICOMLib.xPCFileSystem

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

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

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

Purpose	Remove folder from target computer		
Prototype	<code>long RMDIR(BSTR <i>dirname</i>);</code>		
Member Of	XPCAPICOMLib.xPCFileSystem		
Arguments	<table><tr><td><code>[in] <i>dirname</i></code></td><td>Enter the name of a folder on the target computer.</td></tr></table>	<code>[in] <i>dirname</i></code>	Enter the name of a folder on the target computer.
<code>[in] <i>dirname</i></code>	Enter the name of a folder on the target computer.		
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.		
Description	The xPCFileSystem.RMDIR method removes a folder named <i>dirname</i> from the target computer file system. <i>dirname</i> can be a relative or absolute path name on the target computer.		

xPCFileSystem.ScGetFileName

Purpose	Get name of file for scope
Prototype	BSTR ScGetFileName(long <i>scNum</i>);
Member Of	XPCAPICOMLib.xPCFileSystem
Arguments	[in] <i>scNum</i> Enter the scope number.
Return	Returns the name of the file for the scope.
Description	The xPCFileSystem.ScGetFileName method returns the name of the file to which scope <i>scNum</i> will save signal data.
See Also	API method xPCFileSystem.ScSetFileName

Purpose	Get write mode of file for scope
Prototype	<code>long ScGetWriteMode(long <i>scNum</i>);</code>
Member Of	XPCAPICOMLib.xPCFileSystem
Arguments	[in] <i>scNum</i> Enter the scope number.
Return	This method returns the number indicating the write mode. Values are 0 Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact). 1 Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.
Description	The xPCFileSystem.ScGetWriteMode method returns the write mode of the file for the scope.
See Also	API method xPCFileSystem.ScSetWriteMode

xPCFileSystem.ScGetWriteSize

Purpose Get block write size of data chunks

Prototype `long ScGetWriteSize(long scNum);`

Member Of XPCAPICOMLib.xPCFileSystem

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

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

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

See Also API method xPCFileSystem.ScSetWriteSize

Purpose	Specify file name to contain signal data				
Prototype	<code>long ScSetFileName(long <i>scNum</i>, BSTR <i>filename</i>);</code>				
Member Of	XPCAPICOMLib.xPCFileSystem				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>filename</i></td><td>Enter the name of a file to contain the signal data.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>filename</i>	Enter the name of a file to contain the signal data.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>filename</i>	Enter the name of a file to contain the signal data.				
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.				
Description	The <code>xPCFileSystem.ScSetFileName</code> method sets the name of the file to which the scope will save the signal data. The xPC Target software creates this file in the target computer file system. Note that you can only call this method when the scope is stopped.				
See Also	API method <code>xPCFileSystem.ScGetFileName</code>				

xPCFileSystem.ScSetWriteMode

Purpose Specify when file allocation table entry is updated

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

Member Of XPCAPICOMLib.xPCFileSystem

Arguments

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

Return If the method detects an error, it returns -1. Otherwise, the method returns 0.

Description The `xPCFileSystem.ScSetWriteMode` method specifies when a file allocation table (FAT) entry is updated. Both modes write the signal data to the file, as follows:

- 0 Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
- 1 Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.

See Also API method `xPCFileSystem.ScSetWriteMode`
Scope object property `Mode`

Purpose	Specify that memory buffer collect data in multiples of write size				
Prototype	<code>long ScSetWriteSize(long <i>scNum</i>, long <i>writeSize</i>);</code>				
Member Of	XPCAPICOMLib.xPCFileSystem				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>writeSize</i></td><td>Enter the block size, in bytes, of the data chunks.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>writeSize</i>	Enter the block size, in bytes, of the data chunks.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>writeSize</i>	Enter the block size, in bytes, of the data chunks.				
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.				
Description	The <code>xPCFileSystem.ScSetWriteSize</code> method specifies that a memory buffer collect data in multiples of <i>writeSize</i> . 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. <i>writeSize</i> must be a multiple of 512.				
See Also	API method <code>xPCFileSystem.ScGetWriteSize</code> Scope object property <code>WriteSize</code>				

xPCFileSystem.WriteFile

Purpose	Write to file on target computer						
Prototype	<code>long WriteFile(long <i>fileHandle</i>, long <i>numbytes</i>, VARIANT <i>buffer</i>);</code>						
Member Of	XPCAPICOMLib.xPCFileSystem						
Arguments	<table><tr><td>[in] <i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr><tr><td>[in] <i>numbytes</i></td><td>Enter the number of bytes this method is to write into the file.</td></tr><tr><td>[in] <i>buffer</i></td><td>The contents to write to <i>fileHandle</i> are stored in <i>buffer</i>.</td></tr></table>	[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.	[in] <i>numbytes</i>	Enter the number of bytes this method is to write into the file.	[in] <i>buffer</i>	The contents to write to <i>fileHandle</i> are stored in <i>buffer</i> .
[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.						
[in] <i>numbytes</i>	Enter the number of bytes this method is to write into the file.						
[in] <i>buffer</i>	The contents to write to <i>fileHandle</i> are stored in <i>buffer</i> .						
Return	If the method detects an error, it returns -1. Otherwise, the method returns 0.						
Description	The xPCFileSystem.WriteFile method writes the contents of the VARIANT <i>buffer</i> , of type Byte, to the file specified by <i>fileHandle</i> on the target computer. The <i>fileHandle</i> parameter is the handle of a file previously opened by xPCFSOpenFile. <i>numbytes</i> is the number of bytes to write to the file.						
See Also	API methods xPCFileSystem.CloseFile, xPCFileSystem.GetFileSize, xPCFileSystem.OpenFile, xPCFileSystem.ReadFile						

Purpose	Close RS-232 or TCP/IP communication connection
Prototype	long Close();
Member Of	XPCAPICOMLib.xPCProtocol
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The xPCProtocol.Close method closes the communication channel opened by xPCProtocol.RS232Connect or xPCProtocol.TcpIpConnect.

Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

xPCProtocol.GetLoadTimeOut

Purpose	Return current timeout value for target application initialization
Prototype	<code>long GetLoadTimeOut();</code>
Member Of	<code>XPCAPICOMLib.xPCProtocol</code>
Return	If the method detects an error, it returns -1. Otherwise, it returns the number of seconds allowed for the initialization of the target application.
Description	<p>The <code>xPCProtocol.GetLoadTimeOut</code> method returns the number of seconds allowed for the initialization of the target application.</p> <p>When you load a new target application onto the target computer, the method <code>xPCTarget.LoadApp</code> waits for a certain amount of time before checking to see whether the initialization of the target application is complete. In the case where initialization of the target application is not complete, the method <code>xPCTarget.LoadApp</code> returns a timeout error. By default, <code>xPCTarget.LoadApp</code> checks five times to see whether the target application is ready, with each attempt taking about 1 second. However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout is generated. The method <code>xPCProtocol.SetLoadTimeOut</code>/<code>xPCProtocol.SetLoadTimeOut</code> sets the timeout to a different number.</p> <p>Use the <code>xPCProtocol.GetLoadTimeOut</code> method if you suspect that the current number of seconds (the timeout value) is too short. Then use the <code>xPCProtocol.SetLoadTimeOut</code> method to set the timeout to a higher number.</p>

Purpose	Return error string
Prototype	BSTR GetxPCErrorMsg();
Member Of	XPCAPICOMLib.xPCProtocol
Return	If the xPCProtocol.GetxPCErrorMsg method completes without detecting an error, it returns the string for the last reported error.
Description	The xPCProtocol.GetxPCErrorMsg method returns the string of the last error reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCProtocol.isxPCError method, which detects that an error has occurred.
See Also	API function xPCProtocol.isxPCError

xPCProtocol.Init

Purpose	Initialize xPC Target API DLL
Prototype	<code>long Init();</code>
Member Of	<code>XPCAPICOMLib.xPCProtocol</code>
Return	If the xPC Target DLL, <code>xpcapi.dll</code> loads without causing <code>xPCProtocol.Init</code> to detect an error, the method returns 0. If <code>xpcapi.dll</code> fails to load, this method returns -1.
Description	<p>The <code>xPCProtocol.Init</code> method initializes the xPC Target API by loading the xPC Target DLL, <code>xpcapi.dll</code>, into memory. To load <code>xpcapi.dll</code> into memory, the method requires that the <code>xpcapi.dll</code> file be in one of the following folders:</p> <ul style="list-style-type: none">• The folder in which the application is loaded• The current folder• The Windows system folder

Purpose	Return error status
Prototype	<code>long isxPCError();</code>
Member Of	XPCAPICOMLIB.xPCProtocol
Return	If an error occurred, the method returns 1. Otherwise, it returns 0.
Description	Use the <code>xPCProtocol.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCProtocol</code> class methods. If the method detects that an error occurred, call the <code>xPCProtocol.GetxPCErrorMsg</code> to get the string for the error.
See Also	API function <code>xPCProtocol.GetxPCErrorMsg</code>

xPCProtocol.Port

Purpose	Contain communication channel index
Prototype	<code>long Port();</code>
Member Of	XPCAPICOMLIB.xPCProtocol
Return	If the method detects an error, it returns a nonpositive number. Otherwise, it returns a positive number (the communication channel index).
Description	The <code>xPCProtocol.Port</code> property contains the communication channel index if connection with the target computer succeeds. Note that you only need to use this property when working with a model-specific COM library that you generate from a Simulink model. See “Model-Specific COM Interface Library (model_nameCOMiface.dll)” on page 4-17.

Purpose	Reboot target computer
Prototype	<code>long Reboot();</code>
Member Of	<code>XPCAPICOMLib.xPCProtocol</code>
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The <code>xPCProtocol.Reboot</code> method reboots the target computer. This function does not close the connection to the target computer. You should explicitly close the connection, then reestablish the connection once the target computer has rebooted. Use the methods <code>xPCProtocol.RS232Connect</code> or <code>xPCProtocol.TcpIpConnect</code> to reestablish the connection.

xPCProtocol.RS232Connect

Purpose Open RS-232 connection to target computer

Prototype `long RS232Connect(long comport, long baudrate);`

Member Of XPCAPICOMLib.xPCProtocol

Arguments

[in] <i>comport</i>	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
[in] <i>baudrate</i>	<i>baudrate</i> must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Return The `xPCProtocol.RS232Connect` method returns the port value for the connection. If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCProtocol.RS232Connect` method initiates an RS-232 connection to an xPC Target system. It returns the port value for the connection. Be sure to pass this value to every xPC Target API function that requires a port value.

If you enter a value of 0 for *baudrate*, this function sets the baud rate to the default value (115200).

Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

Purpose Change initialization timeout value

Prototype `long SetLoadTimeOut(long timeOut);`

Member Of XPCAPICOMLib.xPCProtocol

Arguments [in] *timeOut* Enter the new initialization timeout value.

Return If the method detects an error, it returns 0. Otherwise, it returns -1. To get the string description for the error, use `xPCProtocol.GetxPCErrorMsg`.

Description The `xPCProtocol.SetLoadTimeOut` method changes the timeout value for initialization. The *timeOut* value is the time the method `xPCTarget.LoadApp` waits to check whether the model initialization for a new application is complete before returning. It enables you to set the number of initialization attempts to be made before signaling a timeout. When a new target application is loaded onto the target computer, the method `xPCTarget.LoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCTarget.LoadApp` returns a timeout error.

By default, `xPCTarget.LoadApp` checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can be generated.

xPCProtocol.TargetPing

Purpose	Ping target computer
Prototype	long TargetPing;
Member Of	XPCAPICOMLIB.xPCProtocol
Return	The xPCProtocol.TargetPing method does not return an error status. This method returns 1 if it reaches the target computer and the computer responds. If the target computer does not respond, the method returns 0.
Description	<p>The xPCProtocol.TargetPing method pings the target computer and returns 1 or 0 depending on whether the target responds or not. Errors such as the inability to connect to the target are ignored.</p> <p>If you are using TCP/IP, note that xPCProtocol.TargetPing will cause the target computer to close the TCP/IP connection. You can use xPCProtocol.TcpIpConnect to reconnect. You can also use this xPCProtocol.TargetPing feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if your host side program crashes).</p>

Purpose	Open TCP/IP connection to target computer	
Prototype	<code>long TcpIpConnect(BSTR <i>TargetIpAddress</i>, BSTR <i>TargetPort</i>);</code>	
Member Of	XPCAPICOMLIB.xPCProtocol	
Arguments	<code>[in] <i>TargetIpAddress</i></code>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".
	<code>[in] <i>TargetPort</i></code>	Enter the associated IP port as a string. For example, "22222".
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.	
Description	The <code>xPCProtocol.TcpIpConnect</code> method opens a connection to the TCP/IP location specified by the IP address. Use this integer as the <i>TargetPort</i> variable in the xPC Target COM API functions that require a port value.	

xPCProtocol.Term

Purpose	Unload xPC Target API DLL from memory
Prototype	<code>long Term();</code>
Member Of	<code>XPCAPICOMLib.xPCProtocol</code>
Return	The <code>xPCProtocol.Term</code> method always returns -1.
Description	The <code>xPCProtocol.Term</code> method unloads the xPC Target API DLL (<code>xpcapi.dll</code>) from memory. You must call this method when you want to terminate your COM API application.

Purpose	Create new file scope		
Prototype	<code>long AddFileScope(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLib.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .
[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .		
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.		
Description	<p>The <code>xPCScopes.AddFileScope</code> method creates a new file scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddFileScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

xPCScopes.AddHostScope

Purpose	Create new host scope		
Prototype	<code>long AddHostScope(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLib.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .
[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .		
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.		
Description	<p>The <code>xPCScopes.AddHostScope</code> method creates a new host scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddHostScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

Purpose	Create new target scope		
Prototype	<code>long AddTargetScope(long <i>scNum</i>);</code>		
Member Of	<code>XPCAPICOMLib.xPCScopes</code>		
Arguments	<table><tr><td><code>[in] <i>scNum</i></code></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	<code>[in] <i>scNum</i></code>	Enter a number for a new scope. Values are 1, 2, 3. . .
<code>[in] <i>scNum</i></code>	Enter a number for a new scope. Values are 1, 2, 3. . .		
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.		
Description	<p>If the method detects an error, it returns 0. The <code>xPCScopes.AddTargetScope</code> method creates a new scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddTargetScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

xPCScopes.GetScopes

Purpose Get and copy list of scope numbers

Prototype VARIANT GetScopes(long *size*);

Member Of XPCAPICOMLib.xPCScopes

Arguments [in] *size* Specify the size of the VARIANT array returned. This argument must be greater than MAX_SCOPES-1. The elements in the array consist of a list of unsorted integers, terminated by -1.

Return The xPCScopes.GetScopes method returns a VARIANT array with elements containing a list of scope numbers from the target application.

Description The xPCScopes.GetScopes method gets a VARIANT array with elements containing a list of scope numbers currently defined for the target application. Specify the size of the VARIANT array returned. This size must be greater than the maximum number of scopes - 1, up to a maximum of 30 scopes. The elements in the array consist of a list of unsorted integers, terminated by -1.

Purpose	Get error string
Prototype	BSTR GetxPCError();
Member Of	XPCAPICOMLib.xPCScopes
Return	The xPCScopes.GetxPCError method returns the string for the last reported error. If the software has not reported an error, this method returns 0.
Description	The xPCScopes.GetxPCError method gets the string of the last reported error by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCScopes.isxPCError method, which detects that an error has occurred.
See Also	API function xPCScopes.isxPCError

xPCScopes.Init

Purpose	Initialize scope object to communicate with target computer		
Prototype	<code>long Init(IxPCProtocol* xPCProtocol);</code>		
Member Of	XPCAPICOMLib.xPCScopes		
Arguments	<table><tr><td>[in] xPCProtocol</td><td>Specify the communication port of the target computer object for which the scope is to be initialized.</td></tr></table>	[in] xPCProtocol	Specify the communication port of the target computer object for which the scope is to be initialized.
[in] xPCProtocol	Specify the communication port of the target computer object for which the scope is to be initialized.		
Return	If the xPCScopes.Init method initializes the scope object without detecting an error, it returns 0. If the scope object fails to initialize, the method returns -1.		
Description	The xPCScopes.Init method initializes the scope object to communicate with the target computer referenced by the xPCProtocol object.		

Purpose	Get data acquisition status for scope		
Prototype	<code>long IsScopeFinished(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	If the method detects an error, it returns -1. If a scope finishes a data acquisition cycle, this method returns 1. If the scope is in the process of acquiring data, this method returns 0.		
Description	The <code>xPCScopes.IsScopeFinished</code> method gets a 1 or 0 depending on whether scope <i>scNum</i> is finished (state of <code>SCST_FINISHED</code>) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state.		

xPCScopes.isxPCError

Purpose	Get error status
Prototype	<code>long isxPCError();</code>
Member Of	XPCAPICOMLIB.xPCScopes
Return	If an error occurred, the method returns 1. Otherwise, it returns 0.
Description	Use the <code>xPCScopes.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCScopes</code> class methods. If the software detects that an error occurred, call the <code>xPCScopes.GetxPCError</code> method to get the string for the error.
See Also	API function <code>xPCScopes.GetxPCError</code>

Purpose	Remove scope		
Prototype	<code>long RemScope(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.		
Description	The <code>xPCScopes.RemScope</code> method removes the scope with number <i>scNum</i> . Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, use <code>xPCScopes.GetScopes</code> .		

xPCScopes.ScopeAddSignal

Purpose Add signal to scope

Prototype `long ScopeAddSignal(long scNum, long sigNum);`

Member Of XPCAPICOMLib.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>sigNum</i>	Enter a signal number.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeAddSignal` method adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScopes.ScopeGetSignals` to get a list of the signals already present. Use the `xPCTarget.GetSignalIdx` method to get the signal number.

Purpose	Scope autorestart value		
Prototype	<code>long ScopeGetAutoRestart(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	The <code>xPCScopes.ScopeGetAutoRestart</code> method returns the scope autorestart flag value (1 if enabled, 0 if disabled). If the method detects an error, it returns -1.		
Description	The <code>xPCScopes.ScopeGetAutoRestart</code> method gets the autorestart flag value for scope <i>scNum</i> . Autorestart flag can be disabled (0) or enabled (1).		

xPCScopes.ScopeGetData

Purpose Copy scope data to array

Prototype VARIANT ScopeGetData(long *scNum*, long *signal_id*, long *start*, long *numsamples*, long *decimation*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>signal_id</i>	Enter a signal number. Enter -1 to get time stamped data.
[in] <i>start</i>	Enter the first sample from which data retrieval is to start.
[in] <i>numsamples</i>	Enter the number of samples retrieved with a decimation of <i>decimation</i> , starting from the <i>start</i> value.
[in] <i>decimation</i>	Enter a value such that every <i>decimation</i> sample is retrieved in a scope window.

Return The xPCScopes.ScopeGetData method returns a VARIANT array with elements containing the data used in a scope.

Description The xPCScopes.ScopeGetData method gets the data used in a scope. Use this function for scopes of type SCTYPE_HOST. The scope must be either in state Finished or in state Interrupted for the data to be retrievable. (Use the xPCScopes.ScopeGetState method to check the state of the scope.) The data must be retrieved one signal at a time. The calling function determines and allocates the space ahead of time to store the scope data. Use the method xPCScopes.ScopeGetSignals to get the list of signals in the scope for *signal_id*.

To get time stamped data, specify -1 for `signal_id`. From the output, you can then get the number of nonzero elements.

xPCScopes.ScopeGetDecimation

Purpose Get decimation of scope

Prototype `long ScopeGetDecimation(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.ScopeGetDecimation method returns the decimation of scope *scNum*. If the method detects an error, it returns -1.

Description The xPCScopes.ScopeGetDecimation method gets the decimation of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window.

xPCScopes.ScopeGetNumPrePostSamples

Purpose	Get number of pre- or posttriggering samples before triggering scope		
Prototype	<code>long ScopeGetNumPrePostSamples(long scNum);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	The <code>xPCScopes.ScopeGetNumPrePostSamples</code> method returns the number of samples for pre- or posttriggering for scope <i>scNum</i> . If an error occurs, this method returns -1.		
Description	The <code>xPCScopes.ScopeGetNumPrePostSamples</code> method gets the number of samples for pre- or posttriggering for scope <i>scNum</i> . A negative number implies pretriggering, whereas a positive number implies posttriggering samples.		

xPCScopes.ScopeGetNumSamples

Purpose Get number of samples in one data acquisition cycle

Prototype `long ScopeGetNumSamples(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetNumSamples` method returns the number of samples in the scope *scNum*. If the method detects an error, it returns -1.

Description The `xPCScopes.ScopeGetNumSamples` method gets the number of samples in one data acquisition cycle for scope *scNum*.

Purpose Get list of signals

Prototype VARIANT ScopeGetSignals(long *scNum*, long *size*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>size</i>	Enter an integer to allocate the number of elements to be returned in the VARIANT array. This size is required for the method to copy the list of signals into the VARIANT array. The maximum number of signals is 10.

Return The xPCScopes.ScopeGetSignals method returns a VARIANT array with elements consisting of the list of signals defined for a scope.

Description The xPCScopes.ScopeGetSignals method gets the list of signals defined for scope *scNum*. You can use the constant MAX_SIGNALS.

xPCScopes.ScopeGetStartTime

Purpose Get last data acquisition cycle start time

Prototype `double ScopeGetStartTime(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetStartTime` method returns the start time for the last data acquisition cycle of a scope. If the method detects an error, it returns -1.

Description The `xPCScopes.ScopeGetStartTime` method gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`.

Purpose Get state of scope

Prototype BSTR ScopeGetState(long *scNum*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.ScopeGetState method returns the state of scope *scNum*. If the method detects an error, it returns -1.

Description The xPCScopes.ScopeGetState method gets the state of scope *scNum*, or -1 upon error.

Constants to find the scope state have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

xPCScopes.ScopeGetState

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

xPCScopes.ScopeGetTriggerLevel

Purpose Get trigger level for scope

Prototype `double ScopeGetTriggerLevel(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerLevel` method returns the scope trigger level. If the method detects an error, it returns -1.

Description The `xPCScopes.ScopeGetTriggerLevel` method gets the trigger level for scope *scNum*.

xPCScopes.ScopeGetTriggerMode

Purpose Get trigger mode for scope

Prototype `long ScopeGetTriggerMode(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerMode` method returns the scope trigger mode. If the method detects an error, it returns -1.

Description The `xPCScopes.ScopeGetTriggerMode` method gets the trigger mode for scope *scNum*. Use the constants here to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also

API function `xPCScopes.ScopeGetTriggerModeStr`

xPCScopes.ScopeGetTriggerModeStr

Purpose Get trigger mode as string

Prototype `BSTR ScopeGetTriggerModeStr(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerModeStr` method returns a string containing the trigger mode string.

Description The `xPCScopes.ScopeGetTriggerModeStr` method gets the trigger mode string for scope *scNum*. This method returns one of the following strings.

Constant	Description
FreeRun	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
Software	Only user intervention can trigger the scope. No other triggering is possible.
Signal	The scope is triggered only after a signal has crossed a value.
Scope	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

See Also API function `xPCScopes.ScopeGetTriggerMode`

Purpose	Get sample number for triggering scope		
Prototype	<code>long ScopeGetTriggerSample(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	The <code>xPCScopes.ScopeGetTriggerSample</code> method returns a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the method detects an error, it returns -1.		
Description	The <code>xPCScopes.ScopeGetTriggerSample</code> method gets the number of samples a triggering scope (<i>scNum</i>) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope.		

xPCScopes.ScopeGetTriggerSignal

Purpose Get trigger signal for scope

Prototype `long ScopeGetTriggerSignal(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.ScopeGetTriggerSignal method returns the scope trigger signal. If the method detects an error, it returns -1.

Description The xPCScopes.ScopeGetTriggerSignal method gets the trigger signal for scope *scNum*.

Purpose Get trigger slope for scope

Prototype `long ScopeGetTriggerSlope(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerSlope` method returns the scope trigger slope. If the method detects an error, it returns -1.

Description The `xPCScopes.ScopeGetTriggerSlope` method gets the trigger slope of scope *scNum*. Use the constants here to interpret the trigger slope:

String	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger slope must be rising when the signal crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger slope must be falling when the signal crosses the trigger value.

See Also API function `xPCScopes.ScopeGetTriggerSlopeStr`

xPCScopes.ScopeGetTriggerSlopeStr

Purpose Get trigger slope as string

Prototype `BSTR ScopeGetTriggerSlopeStr(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The `xPCScopes.ScopeGetTriggerSlopeStr` method returns a string containing the trigger slope string.

Description The `xPCScopes.ScopeGetTriggerSlopeStr` method gets the trigger slope string for scope *scNum*. This method returns one of the following strings:

String	Description
Either	The trigger slope can be either rising or falling.
Rising	The trigger slope must be rising when the signal crosses the trigger value.
Falling	The trigger slope must be falling when the signal crosses the trigger value.

See Also API function `xPCScopes.ScopeGetTriggerSlope`

Purpose Get type of scope

Prototype BSTR ScopeGetType(long *scNum*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.ScopeGetType method returns the scope type as a string. If the method detects an error, it returns -1.

Description The xPCScopes.ScopeGetType method gets the type of scope *scNum*. This method returns one of the following strings:

String	Description
HOST	Host scope
Target	Target scope

xPCScopes.ScopeRemSignal

Purpose Remove signal from scope

Prototype `long ScopeRemSignal(long scNum, long sigNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>sigNum</i>	Enter a signal number.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeRemSignal` method removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCScopes.GetScopes` to determine the existing scopes, and use `xPCScopes.ScopeGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

Purpose	Scope autorestart value				
Prototype	<code>long ScopeSetAutoRestart(long <i>scNum</i>, long <i>onoff</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>onoff</i></td><td>Enter value to enable (1) or disable (0) scope autorestart.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>onoff</i>	Enter value to enable (1) or disable (0) scope autorestart.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>onoff</i>	Enter value to enable (1) or disable (0) scope autorestart.				
Return	The <code>xPCScopes.ScopeSetAutoRestart</code> method returns the scope autorestart flag value (1 if enabled, 0 if disabled). If the method detects an error, it returns -1.				
Description	The <code>xPCScopes.ScopeSetAutoRestart</code> method sets the autorestart flag value for scope <i>scNum</i> . Autorestart flag can be disabled (0) or enabled (1).				

xPCScopes.ScopeSetDecimation

Purpose	Set decimation of scope				
Prototype	<code>long ScopeSetDecimation(long <i>scNum</i>, long <i>decimation</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>decimation</i></td><td>Enter an integer for the decimation.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>decimation</i>	Enter an integer for the decimation.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>decimation</i>	Enter an integer for the decimation.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCScopes.ScopeSetDecimation</code> method sets the <i>decimation</i> of scope <i>scNum</i> . The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use this function only when the scope is stopped. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope.				

xPCScopes.ScopeSetNumPrePostSamples

Purpose Set number of pre- or posttriggering samples before triggering scope

Prototype `long ScopeSetNumPrePostSamples(long scNum, long prepost);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>prepost</i>	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeSetNumPrePostSamples` method sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scope numbers.

xPCScopes.ScopeSetNumSamples

Purpose Set number of samples in one data acquisition cycle

Prototype `long ScopeSetNumSamples(long scNum, long samples);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>samples</i>	Enter the number of samples you want to acquire in one cycle.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeSetNumSamples` method sets the number of samples for scope *scNum* to *samples*. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

xPCScopes.ScopeSetTriggerLevel

Purpose	Set trigger level for scope				
Prototype	<code>long ScopeSetTriggerLevel(long <i>scNum</i>, double <i>level</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>level</i></td><td>Value for a signal to trigger data acquisition with a scope.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>level</i>	Value for a signal to trigger data acquisition with a scope.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>level</i>	Value for a signal to trigger data acquisition with a scope.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCScopes.ScopeSetTriggerLevel</code> method sets the trigger level to <i>level</i> for scope <i>scNum</i> . Use this function only when the scope is stopped. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope.				

xPCScopes.ScopeSetTriggerMode

Purpose Set trigger mode of scope

Prototype `long ScopeSetTriggerMode(long scNum, long triggermode);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
[in] <i>triggermode</i>	Trigger mode for a scope.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeSetTriggerMode` method sets the trigger mode of scope *scNum* to *triggermode*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes. Use the constants defined here to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.

xPCScopes.ScopeSetTriggerMode

Constant	Value	Description
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code>).

xPCScopes.ScopeSetTriggerSample

Purpose	Set sample number for triggering scope				
Prototype	<code>long ScopeSetTriggerSample(long scNum, long trigScSample);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>trigScSample</i></td><td>Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>trigScSample</i>	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>trigScSample</i>	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	<p>The <code>xPCScopes.ScopeSetTriggerSample</code> method sets the number of samples (<i>trigScSample</i>) a triggering scope acquires before it triggers a second scope (<i>scNum</i>). Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.</p> <p>For meaningful results, set <i>trigScSample</i> between -1 and (<i>nSamp</i>-1). <i>nSamp</i> 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.</p> <p>If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, use a value of -1 for <i>trigScSamp</i>.</p>				

xPCScopes.ScopeSetTriggerSignal

Purpose	Select signal to trigger scope				
Prototype	<code>long ScopeSetTriggerSignal(long <i>scNum</i>, long <i>triggerSignal</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>triggerSignal</i></td><td>Enter a signal number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>triggerSignal</i>	Enter a signal number.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>triggerSignal</i>	Enter a signal number.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCScopes.ScopeSetTriggerSignal</code> method sets the trigger signal of scope <i>scNum</i> to <i>trigSig</i> . The trigger signal <i>trigSig</i> must be one of the signals in the scope. Use this method only when the scope is stopped. You can use <code>xPCScopes.ScopeGetSignals</code> to get the list of signals in the scope. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.				

xPCScopes.ScopeSetTriggerSlope

Purpose Set slope of signal that triggers scope

Prototype `long ScopeSetTriggerSlope(long scNum, long triggerSlope);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments

- [in] *scNum* Enter the scope number.
- [in] *triggerSlope* Enter the slope mode for the signal that triggers the scope.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeSetTriggerSlope` method sets the trigger slope of scope *scNum* to *triggerSlope*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Use the constants defined here to set the trigger slope:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger signal value must be rising when it crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger signal value must be falling when it crosses the trigger value.

Purpose	Set software trigger of scope
Prototype	<code>long ScopeSoftwareTrigger(long scNum);</code>
Member Of	XPCAPICOMLIB.xPCScopes
Arguments	[in] <i>scNum</i> Enter the scope number.
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	<p>The <code>xPCScopes.ScopeSoftwareTrigger</code> method triggers scope <i>scNum</i>. The scope must be in the state <code>Waiting for trigger</code> for this method to succeed. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.</p> <p>You can use the <code>xPCScopes.ScopeSoftwareTrigger</code> method to trigger the scope, regardless of the trigger mode.</p>

xPCScopes.ScopeStart

Purpose Start data acquisition for scope

Prototype `long ScopeStart(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeStart` method starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call `xPCScopes.ScopeGetState` to check the state of the scope or, for host scopes that are already started, call `xPCScopes.IsScopeFinished`. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose Stop data acquisition for scope

Prototype `long ScopeStop(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCScopes.ScopeStop` method stops the scope *scNum*. This sets the scope to the `Interrupted` state. The scope must be running for this function to succeed. Use `xPCScopes.ScopeGetState` to determine the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

xPCScopes.TargetScopeGetGrid

Purpose Get status of grid line for particular scope

Prototype `long TargetScopeGetGrid(long scNum);`

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.TargetScopeGetGrid method returns the state of the grid lines for scope *scNum*. If the method detects an error, it returns -1.

Description The xPCScopes.TargetScopeGetGrid method gets the state of the grid lines for scope *scNum* (which must be of type SCTYPE_TARGET). A return value of 1 implies grid on, while 0 implies grid off. Note that when the scope mode is set to SCMODE_NUMERICAL, the grid is not drawn even when the grid mode is set to 1.

Tip

- Use the xPCScopes.GetScopes method to get a list of scopes.
 - Use xPCScopes.TargetScopeGetMode and xPCScopes.TargetScopeSetMode to retrieve and set the scope mode.
-

Purpose	Get scope mode for displaying signals		
Prototype	<code>long TargetScopeGetMode(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	<p>The <code>xPCScopes.TargetScopeGetMode</code> method returns the value corresponding to the scope mode. The possible values are</p> <ul style="list-style-type: none">• <code>SCMODE_NUMERICAL = 0</code>• <code>SCMODE_REDRAW = 1</code>• <code>SCMODE_SLIDING = 2</code>• <code>SCMODE_ROLLING = 3</code> <p>If the method detects an error, it returns -1.</p>		
Description	The <code>xPCScopes.TargetScopeGetMode</code> method gets the mode of the scope <i>scNum</i> , which must be of type <code>SCTYPE_TARGET</code> . Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.		
See Also	API function <code>xPCScopes.TargetScopeGetModeStr</code>		

xPCScopes.TargetScopeGetModeStr

Purpose Get scope mode string for displaying signals

Prototype BSTR TargetScopeGetModeStr(long *scNum*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments [in] *scNum* Enter the scope number.

Return The xPCScopes.TargetScopeGetModeStr method returns the string corresponding to the scope mode. The possible strings are

- Numerical
- Redraw
- Sliding
- Rolling

Description The xPCScopes.TargetScopeGetModeStr method gets the mode string of the scope *scNum*, which must be of type SCTYPE_TARGET. Use the xPCScopes.GetScopes method to get a list of scopes.

See Also API function xPCScopes.TargetScopeGetMode

xPCScopes.TargetScopeGetViewMode

Purpose	Get view mode for target computer display
Prototype	<code>long TargetScopeGetViewMode();</code>
Member Of	XPCAPICOMLIB.xPCScopes
Return	The <code>xPCScopes.TargetScopeGetViewMode</code> method returns the view mode for the target computer screen. If the method detects an error, it returns -1.
Description	The <code>xPCScopes.TargetScopeGetViewMode</code> method gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is of the scope currently displayed on the screen. If the value is 0, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).

xPCScopes.TargetScopeGetYLimits

Purpose	Get <i>y</i> -axis limits for scope
Prototype	VARIANT TargetScopeGetYLimits(long <i>scNum</i>);
Member Of	XPCAPICOMLIB.xPCScopes
Arguments	[in] <i>scNum</i> Enter the scope number.
Return	The xPCScopes.TargetScopeGetYLimits method returns the upper and lower limits for target scopes.
Description	The xPCScopes.TargetScopeGetYLimits method gets and copies the upper and lower limits for a scope of type SCTYPE_TARGET and with scope number <i>scNum</i> . If both elements are zero, the limits are autoscaled. Use the xPCScopes.GetScopes method to get a list of scopes.

Purpose	Set grid mode for scope				
Prototype	<code>long TargetScopeSetGrid(long <i>scNum</i>, long <i>gridonoff</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>gridonoff</i></td><td>Enter a grid value (0 or 1).</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>gridonoff</i>	Enter a grid value (0 or 1).
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>gridonoff</i>	Enter a grid value (0 or 1).				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCScopes.TargetScopeSetGrid</code> method sets the grid of a scope of type <code>SCTYPE_TARGET</code> and scope number <i>scNum</i> to <i>gridonoff</i> . If <i>gridonoff</i> is 0, the grid is off. If <i>gridonoff</i> is 1, the grid is on and grid lines are drawn on the scope window. When the drawing mode of scope <i>scNum</i> is set to <code>SCMODE_NUMERICAL</code> , the grid is not drawn even when the grid mode is set to 1. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.				

xPCScopes.TargetScopeSetMode

Purpose Set display mode for scope

Prototype long TargetScopeSetMode(long *scNum*, long *mode*);

Member Of XPCAPICOMLIB.xPCScopes

Arguments

[in] <i>scNum</i>	Enter the scope number.
in] <i>mode</i>	Enter the value for the mode.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The xPCScopes.TargetScopeSetMode method sets the mode of a scope of type SCTYPE_TARGET and scope number *scNum* to *mode*. You can use one of the following constants for *mode*:

- SCMODE_NUMERICAL = 0
- SCMODE_REDRAW = 1
- SCMODE_SLIDING = 2
- SCMODE_ROLLING = 3

Use the xPCScopes.GetScopes method to get a list of scopes.

xPCScopes.TargetScopeSetViewMode

Purpose	Set view mode for scope		
Prototype	<code>long TargetScopeSetViewMode(long <i>scNum</i>);</code>		
Member Of	XPCAPICOMLIB.xPCScopes		
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.		
Description	The <code>xPCScopes.TargetScopeSetViewMode</code> method sets the target computer screen to display one scope with scope number <i>scNum</i> . If you set <i>scNum</i> to 0, the target computer screen displays all the defined scopes. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.		

xPCScopes.TargetScopeSetYLimits

Purpose	Set <i>y</i> -axis limits for scope				
Prototype	<code>long TargetScopeSetYLimits(long <i>scNum</i>, SAFEARRAY(double)* <i>YLimitarray</i>);</code>				
Member Of	XPCAPICOMLIB.xPCScopes				
Arguments	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in, out] <i>YLimitarray</i></td><td>Enter a two-element array.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in, out] <i>YLimitarray</i>	Enter a two-element array.
[in] <i>scNum</i>	Enter the scope number.				
[in, out] <i>YLimitarray</i>	Enter a two-element array.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCScopes.TargetScopeSetYLimits</code> method sets the <i>y</i> -axis limits for a scope with scope number <i>scNum</i> and type <code>SCTYPE_TARGET</code> to the values in the double array <i>YLimitArray</i> . 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 <code>xPCScopes.GetScopes</code> method to get a list of scopes.				

Purpose	Get average task execution time
Prototype	<code>double AverageTET();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.AverageTET</code> method returns the average task execution time (TET) for the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.AverageTET</code> method gets the TET for the target application. You can use this function when the target application is running or when it is stopped.

xPCTarget.GetAppName

Purpose	Get target application name
Prototype	BSTR GetAppName();
Member Of	XPCAPICOMLib.xPCTarget
Return	The xPCTarget.GetAppName method returns a string with the name of the target application.
Description	The xPCTarget.GetAppName method gets the name of the target application. You can use the return value, <i>model_name</i> , in a printf or similar statement. In case of error, the string is unchanged. Be sure to allocate enough space to accommodate the longest target name you have.

Purpose	Get execution time for target application
Prototype	<code>double GetExecTime();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.GetExecTime</code> method returns the current execution time for a target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetExecTime</code> method gets the current execution time for the running target application. If the target application is stopped, the value is the last running time when the target application was stopped. If the target application is running, the value is the current running time.

xPCTarget.GetNumOutputs

Purpose	Get number of outputs
Prototype	<code>long GetNumOutputs();</code>
Member Of	<code>XPCAPICOMLib.xPCTarget</code>
Return	The <code>xPCTarget.GetNumOutputs</code> method returns the number of outputs in the current target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetNumOutputs</code> method gets the number of outputs in the target application. The number of outputs equals the sum of the input signal widths of the output blocks at the root level of the Simulink model.

Purpose	Get number of tunable parameters
Prototype	<code>long GetNumParams();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.GetNumParams</code> method returns the number of tunable parameters in the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetNumParams</code> method gets the number of tunable parameters in the target application. Use this method to see how many parameters you can get or modify.

xPCTarget.GetNumSignals

Purpose	Get number of signals
Prototype	<code>long GetNumSignals();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.GetNumSignals</code> method returns the number of signals in the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetNumSignals</code> method gets the total number of signals in the target application that can be monitored from the host. Use this method to see how many signals you can monitor.

Purpose	Get number of states
Prototype	<code>long GetNumStates();</code>
Member Of	<code>XPCAPICOMLib.xPCTarget</code>
Return	The <code>xPCTarget.GetNumStates</code> method returns the number of states in the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetNumStates</code> method gets the number of states in the target application.

xPCTarget.GetOutputLog

Purpose Copy output log data to array

Prototype VARIANT GetOutputLog(long *start*, long *numsamples*, long *decimation*, long *output_id*);

Member Of XPCAPICOMLib.xPCTarget

Arguments

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the output log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>output_id</i>	Enter an output identification number.

Return The xPCTarget.GetOutputLog method returns output log data. You get the data for each output signal. If the method detects an error, it returns VT_ERROR, a scalar.

Description The xPCTarget.GetOutputLog method gets the output log and copies that log to an array. Output IDs range from 0 to (N-1), where N is the return value of xPCTarget.GetNumOutputs. Entering 1 for *decimation* copies all values. Entering N copies every Nth value.

For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Get the maximum number of samples by calling the method xPCTarget.NumLogSamples.

Note that the target application must be stopped before you get the output log data.

Purpose	Get parameter values
Prototype	VARIANT GetParam(long <i>paramIdx</i>);
Member Of	XPCAPICOMLib.xPCTarget
Arguments	[in] <i>paramIdx</i> Enter the index for a parameter.
Return	The xPCTarget.GetParam method returns the parameter values of a parameter.
Description	The xPCTarget.GetParam method gets the parameter values of a parameter identified by <i>paramIdx</i> . This method returns an array of type VARIANT containing the parameter values, with the conversion of the values being done in column-major format. Each element in the array is a double, regardless of the data type of the actual parameter. You can query the dimensions of the array by calling the method xPCTarget.GetParamDims. See the Microsoft Visual Basic .NET 2003 solution located in <code>C:\matlabroot\toolbox\rtw\targets\xpc\api\VBNET\SigsAndParamsDemo</code> for an example of how to use this method.
See Also	API method xPCTarget.GetParamDims, xPCTarget.SetParam

xPCTarget.GetParamDims

Purpose Get row and column dimensions of parameter

Prototype VARIANT GetParamDims(long *paramIdx*);

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *paramIdx* Parameter index.

Return The xPCTarget.GetParamDims method returns a VARIANT array of two elements.

Description The xPCTarget.GetParamDims method gets a VARIANT array of two elements. The first element contains the number of rows of the parameter, the second element contains the number of columns for your parameter.

Purpose	Get parameter index				
Prototype	<code>long GetParamIdx(BSTR <i>blockName</i>, BSTR <i>paramName</i>);</code>				
Member Of	XPCAPICOMLib.xPCTarget				
Arguments	<table><tr><td>[in] <i>blockName</i></td><td>Enter the full block path generated by the Simulink Coder software.</td></tr><tr><td>[in] <i>paramName</i></td><td>Enter the parameter name for a parameter associated with the block.</td></tr></table>	[in] <i>blockName</i>	Enter the full block path generated by the Simulink Coder software.	[in] <i>paramName</i>	Enter the parameter name for a parameter associated with the block.
[in] <i>blockName</i>	Enter the full block path generated by the Simulink Coder software.				
[in] <i>paramName</i>	Enter the parameter name for a parameter associated with the block.				
Return	The <code>xPCTarget.GetParamIdx</code> method returns the parameter index for the parameter name. If the method detects an error, it returns -1.				
Description	The <code>xPCTarget.GetParamIdx</code> method gets the parameter index for the parameter name (<i>paramName</i>) associated with a Simulink block (<i>blockName</i>). Both <i>blockName</i> and <i>paramName</i> must be identical to those generated at target application building time. The block names should be referenced from the file <i>model_namept.m</i> in the generated code, where <i>model_name</i> is the name of the model. Note that a block can have one or more parameters.				

xPCTarget.GetParamName

Purpose Get parameter name

Prototype VARIANT GetParamName(long *paramIdx*);

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *paramIdx* Enter a parameter index.

Return The xPCTarget.GetParamName method returns a VARIANT array that contains two elements, the block path and parameter name, as strings.

Description The xPCTarget.GetParamName method gets the parameter name and block name for a parameter with the index *paramIdx*. If *paramIdx* is invalid, xPCGetLastError returns nonzero, and the strings are unchanged. Get the parameter index with the method xPCTarget.GetParamIdx.

Purpose	Get sample time
Prototype	<code>double GetSampleTime();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.GetSampleTime</code> method returns the sample time, in seconds, of the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetSampleTime</code> method gets the sample time, in seconds, of the target application. You can get the error by using the method <code>xPCGetLastError</code> .

xPCTarget.GetSignal

Purpose Get signal value

Prototype `double GetSignal(long sigNum);`

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *sigNum* Enter a signal number.

Return The `xPCTarget.GetSignal` method returns the current value of signal *sigNum*. If the method detects an error, it returns -1.

Description The `xPCTarget.GetSignal` method gets the current value of a signal. Use the `xPCTarget.GetSignalIdx` method to get the signal number.

xPCTarget.GetSignalidsfromLabel

Purpose	Get signal IDs from signal label
Prototype	VARIANT GetSignalidsfromLabel(BSTR <i>sigLabel</i>);
Member Of	XPCAPICOMLib.xPCTarget
Arguments	[in] <i>sigLabel</i> Enter a signal label.
Return	The xPCTarget.GetSignalidsfromLabel method returns a VARIANT array of the signal elements contained in the signal <i>sigLabel</i> . If no labels exist, the method returns an empty string.
Description	<p>The xPCTarget.GetSignalidsfromLabel method returns a VARIANT array of the signal elements contained in the signal <i>sigLabel</i>. Signal labels must be unique.</p> <p>This method assumes that you have labeled the signal for which you request the indices (see the Signal name parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.</p>
See Also	API method xPCTarget.GetSignalLabel

xPCTarget.GetSignalLabel

Purpose Get signal label

Prototype BSTR GetSignalLabel(long *sigIdx*);

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *sigIdx* Enter a signal index.

Return The xPCTarget.GetSignalLabel method returns the label of the signal. If no labels exist, the method returns an empty string.

Description The xPCTarget.GetSignalLabel method copies and gets the signal label of a signal with *sigIdx*. The method returns the signal label. This method assumes that you already know the signal index. Signal labels must be unique.

This method assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

See Also API method xPCTarget.GetSignalidsfromLabel

Purpose Get signal index

Prototype long GetSignalIdx(BSTR *sigName*);

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *sigName* Enter a signal name.

Return The xPCTarget.GetSignalIdx method returns the index for the signal with name *sigName*. If the method detects an error, it returns -1.

Description The xPCTarget.GetSignalIdx method gets the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file *model_namebio.m* in the generated code, where *model_name* is the name of the model. The creator of the application should already know the signal name.

xPCTarget.GetSignalName

Purpose Copy signal name to character array

Prototype BSTR GetSignalName(long *sigIdx*);

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *sigIdx* Enter a signal index.

Return The xPCTarget.GetSignalName method returns the name of the signal.

Description The xPCTarget.GetSignalName method copies and gets the signal name, including the block path, of a signal with *sigIdx*. The method returns a signal name, which makes it convenient to use in a `printf` or similar statement. This method assumes that you already know the signal index.

Purpose Get vector of signal values

Prototype `VARIANT GetSignals(long NumOfSignals, SAFEARRAY(int)* SignalsIdxArray);`

Member Of XPCAPICOMLib.xPCTarget

Arguments

[in] <i>NumOfSignals</i>	Enter the number of signals to acquire (the number of IDs in <i>SignalsIdxArray</i>).
[out] <i>SignalsIdxArray</i>	Enter the IDs of the signals to acquire.

Return The `xPCTarget.GetSignals` method returns a double-valued variant array containing the current value of a vector of signals. If the method detects an error, it returns `VT_ERROR`, a scalar.

Description This function returns the values of a vector of up to 1000 signals as fast as it can acquire them. The values are converted to doubles regardless of the actual data type of the signal.

Tip

- Pass an integer array of signal numbers into *SignalsIdxArray*. Get the signal numbers with the function `xPCTarget.GetSignalIdx`.
 - The signal values may not be at the same time step. To get signal values at the same time step, define a scope of type `SCTYPE_HOST` and use `xPCScopes.ScopeGetData`.
-

The function `xPCTarget.GetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result.

xPCTarget.GetSignals

However, xPCGetSignals is faster and the signal values are more likely to be spaced closely together.

See Also

API functions `xPCTarget.GetSignal`, `xPCTarget.GetSignalIdx`

Purpose Get width of signal

Prototype `long GetSignalWidth(long sigIdx);`

Member Of XPCAPICOMLib.xPCTarget

Arguments [in] *sigIdx* Enter the index of a signal.

Return The `xPCTarget.GetSignalWidth` method returns the signal width for a signal with *sigIdx*. If the method detects an error, it returns -1.

Description The `xPCTarget.GetSignalWidth` method gets the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector. A signal's width is the number of signals in the vector.

xPCTarget.GetStateLog

Purpose Get state log

Prototype VARIANT GetStateLog(long *start*, long *numsamples*, long *decimation*, long *state_id*);

Member Of XPCAPICOMLib.xPCTarget

Arguments

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the output log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>state_id</i>	Enter a state identification number.
[out, retval] <i>Outarray</i>	The log is stored in <i>Outarray</i> , whose allocation is the responsibility of the caller.

Return The xPCTarget.GetStateLog method returns the state log. If the method detects an error, it returns VT_ERROR, a scalar.

Description The xPCTarget.GetStateLog method gets the state log. You get the data for each state signal in turn by specifying the *state_id*. State IDs range from 1 to (N-1), where N is the return value of xPCTarget.GetNumStates. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

Purpose	Get stop time
Prototype	<code>double GetStopTime();</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	The <code>xPCTarget.GetStopTime</code> method returns the stop time as a double, in seconds, of the target application. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.GetStopTime</code> method gets the stop time, in seconds, of the target application. This is the amount of time the target application runs before stopping.

xPCTarget.GetTETLog

Purpose Get TET log

Prototype VARIANT GetTETLog(long *start*, long *numsamples*, long *decimation*);

Member Of XPCAPICOMLib.xPCTarget

Arguments

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the TET log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[out, retval] <i>Outarray</i>	The log is stored in <i>Outarray</i> , whose allocation is the responsibility of the caller.

Return The xPCTarget.GetTETLog method returns the TET log. If the method detects an error, it returns VT_ERROR, a scalar.

Description The xPCTarget.GetTETLog method gets the task execution time (TET) log. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

Purpose	Get time log						
Prototype	VARIANT GetTimeLog(long <i>start</i> , long <i>numsamples</i> , long <i>decimation</i>);						
Member Of	XPCAPICOMLib.xPCTarget						
Arguments	<table><tr><td>[in] <i>start</i></td><td>Enter the index of the first sample to copy.</td></tr><tr><td>[in] <i>numsamples</i></td><td>Enter the number of samples to copy from the time log.</td></tr><tr><td>[in] <i>decimation</i></td><td>Select whether to copy all the sample values or every Nth value.</td></tr></table>	[in] <i>start</i>	Enter the index of the first sample to copy.	[in] <i>numsamples</i>	Enter the number of samples to copy from the time log.	[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>start</i>	Enter the index of the first sample to copy.						
[in] <i>numsamples</i>	Enter the number of samples to copy from the time log.						
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.						
Return	The xPCTarget.GetTimeLog method returns the time log. If the method detects an error, it returns VT_ERROR, a scalar.						
Description	<p>The xPCTarget.GetTimeLog method gets the time log. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering 1 for <i>decimation</i> copies all values. Entering N copies every Nth value. For <i>start</i>, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the number of samples.</p> <p>Note that the target application must be stopped before you get the number.</p>						

xPCTarget.GetxPCError

Purpose	Get error string
Prototype	BSTR GetxPCError();
Member Of	XPCAPICOMLib.xPCTarget
Return	The xPCTarget.GetxPCError method returns the string for the last reported error. If the software has not reported an error, this method returns 0.
Description	The xPCTarget.GetxPCError method gets the string of the error last reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCTarget.isxPCError method, which detects that an error has occurred.
See Also	API method xPCTarget.isxPCError

Purpose	Initialize target object to communicate with target computer
Prototype	<code>long Init(IxPCProtocol* xPCProtocol);</code>
Member Of	XPCAPICOMLib.xPCTarget
Return	<p>If the method detects an error, it returns -1. Otherwise, it returns 0.</p> <p>If the xPCTarget.Init method initializes the target object without detecting an error, it returns 0. If the target object fails to initialize, this method returns -1.</p>
Description	The xPCTarget.Init method initializes the target object to communicate with the target computer referenced by the xPCProtocol object.

xPCTarget.IsAppRunning

Purpose	Return running status for target application
Prototype	<code>long IsAppRunning();</code>
Member Of	<code>XPCAPICOMLib.xPCTarget</code>
Return	If the target application is stopped, the <code>xPCTarget.IsAppRunning</code> method returns 0. If the target application is running, this method returns 1. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.IsAppRunning</code> method returns 1 or 0 depending on whether the target application is stopped or running.

Purpose	Return overload status for target computer
Prototype	<code>long IsOverloaded();</code>
Member Of	<code>XPCAPICOMLib.xPCTarget</code>
Return	If the target application has overloaded the CPU, the <code>xPCTarget.IsOverloaded</code> method returns 1. If it has not overloaded the CPU, the method returns 0. If the method detects an error, it returns -1.
Description	The <code>xPCTarget.IsOverloaded</code> method checks if the target application has overloaded the target computer and returns 1 if it has and 0 if it has not. If the target application is not running, the method returns 0.

xPCTarget.isxPCError

Purpose	Return error status
Prototype	<code>long isxPCError();</code>
Member Of	XPCAPICOMLIB.xPCTarget
Return	If an error occurred, the method returns 1. Otherwise, it returns 0.
Description	Use the <code>xPCTarget.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCTarget</code> class methods. If the method detects that an error occurred, call the <code>xPCTarget.GetxPCError</code> method to get the string for the error.
See Also	API method <code>xPCTarget.GetxPCError</code>

Purpose Load target application onto target computer

Prototype long LoadApp(BSTR *pathstr*, BSTR *filename*);

Member Of XPCAPICOMLIB.xPCTarget

Arguments

[in] <i>pathstr</i>	Enter the full path to the target application file, excluding the file name. For example, in C, use a string like "C:\\work", in Microsoft Visual Basic, use a string like 'C:\\work'.
[in] <i>filename</i>	Enter the name of a compiled target application (*.dlm) without the file extension. For example, in C use a string like "xpcosc", in Microsoft Visual Basic, use a string like 'xpcosc'.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The xPCTarget.LoadApp method loads the compiled target application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to NULL or to the string 'nopath' if the application is in the current folder. The variable *filename* must not contain the target application extension.

Before returning, xPCTarget.LoadApp waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, xPCTarget.LoadApp returns a timeout error to indicate a connection problem (for example, ETCPREAD). By default, xPCTarget.LoadApp checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can

xPCTarget.LoadApp

be generated. The methods `xPCProtocol.GetLoadTimeOut` and `xPCProtocol.SetLoadTimeOut` control the number of attempts made.

Purpose	Copy maximum task execution time to array
Prototype	VARIANT MaximumTET();
Member Of	XPCAPICOMLIB.xPCTarget
Return	The xPCTarget.MaximumTET method returns a VARIANT object containing the maximum task execution time (TET) and the time at which the maximum TET was achieved. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.
Description	The xPCTarget.MaximumTET method returns the maximum TET that was achieved during the previous target application run.

xPCTarget.MaxLogSamples

Purpose	Return maximum number of samples that can be in log buffer
Prototype	<code>long MaxLogSamples();</code>
Member Of	<code>XPCAPICOMLIB.xPCTarget</code>
Return	The <code>xPCTarget.MaxLogSamples</code> method returns the total number of samples. If the method detects an error, it returns -1.
Description	<p>The <code>xPCTarget.MaxLogSamples</code> method returns the total number of samples that can be returned in the logging buffers.</p> <p>Note that the target application must be stopped before you get the number.</p>

Purpose	Copy minimum task execution time to array
Prototype	VARIANT MinimumTET();
Member Of	XPCAPICOMLIB.xPCTarget
Return	The xPCTarget.MinimumTET method returns a VARIANT object containing the minimum task execution time (TET) and the time at which the minimum TET was achieved. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.
Description	The xPCTarget.MinimumTET method returns the minimum task execution time (TET) that was achieved during the previous target application run.

xPCTarget.NumLogSamples

Purpose	Return number of samples in log buffer
Prototype	<code>long NumLogSamples();</code>
Member Of	XPCAPICOMLIB.xPCTarget
Return	The <code>xPCTarget.NumLogSamples</code> method returns the number of samples in the log buffer. If the method detects an error, it returns -1.
Description	<p>The <code>xPCTarget.NumLogSamples</code> method returns the number of samples in the log buffer. In contrast to <code>xPCTarget.MaxLogSamples</code>, which returns the maximum number of samples that can be logged (because of buffer size constraints), <code>xPCTarget.NumLogSamples</code> returns the number of samples actually logged.</p> <p>Note that the target application must be stopped before you get the number.</p>

Purpose	Return number of times log buffer wraps
Prototype	<code>long NumLogWraps();</code>
Member Of	XPCAPICOMLIB.xPCTarget
Return	The <code>xPCTarget.NumLogWraps</code> method returns the number of times the log buffer wraps. If the method detects an error, it returns -1.
Description	<p>The <code>xPCTarget.NumLogWraps</code> method returns the number of times the log buffer wraps.</p> <p>Note that the target application must be stopped before you get the number.</p>

xPCTarget.SetParam

Purpose	Change parameter value				
Prototype	<code>long SetParam(long <i>paramIdx</i>, SAFEARRAY(double)* <i>newparamVal</i>);</code>				
Member Of	XPCAPICOMLIB.xPCTarget				
Arguments	<table><tr><td>[in] <i>paramIdx</i></td><td>Parameter index.</td></tr><tr><td>[in, out] <i>newparamVal</i></td><td>Vector of doubles, assumed to be the size required by the parameter type.</td></tr></table>	[in] <i>paramIdx</i>	Parameter index.	[in, out] <i>newparamVal</i>	Vector of doubles, assumed to be the size required by the parameter type.
[in] <i>paramIdx</i>	Parameter index.				
[in, out] <i>newparamVal</i>	Vector of doubles, assumed to be the size required by the parameter type.				
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.				
Description	The <code>xPCTarget.SetParam</code> method sets the parameter <i>paramIdx</i> to the value in <i>newparamVal</i> . For matrices, <i>newparamVal</i> should be a vector representation of the matrix in column-major format. Although <i>newparamVal</i> is a vector of doubles, the method converts the values to the expected data types (using truncation) before setting them.				
See Also	API methods <code>xPCTarget.GetParam</code> , <code>xPCTarget.GetParamDims</code> , <code>xPCTarget.GetParamIdx</code>				

Purpose	Change sample time for target application
Prototype	<code>long SetSampleTime(double ts);</code>
Member Of	XPCAPICOMLIB.xPCTarget
Arguments	[in] <i>ts</i> Sample time for the target application.
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The <code>xPCTarget.SetSampleTime</code> method sets the sample time, in seconds, of the target application to <i>ts</i> . Use this method only when the application is stopped.

xPCTarget.SetStopTime

Purpose Change stop time of target application

Prototype `long SetStopTime(double tfinal);`

Member Of XPCAPICOMLIB.xPCTarget

Arguments [in] *tfinal* Enter the stop time, in seconds.

Return If the method detects an error, it returns 0. Otherwise, it returns -1.

Description The `xPCTarget.SetStopTime` method sets the stop time of the target application to the value in *tfinal*. The target application will run for this number of seconds before stopping. Set *tfinal* to -1.0 to set the stop time to infinity.

Purpose	Start target application
Prototype	long StartApp()
Member Of	XPCAPICOMLIB.xPCTarget
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The xPCTarget.StartApp method starts the target application loaded on the target machine.

xPCTarget.StopApp

Purpose	Stop target application
Prototype	<code>long StopApp();</code>
Member Of	<code>XPCAPICOMLIB.xPCTarget</code>
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The <code>xPCTarget.StopApp</code> method stops the target application loaded on the target computer. The target application remains loaded, and the parameter changes you made remain intact. If you want to stop and unload an application, use <code>xPCTarget.UnLoadApp</code> .

Purpose	Unload target application
Prototype	<code>long UnLoadApp();</code>
Member Of	XPCAPICOMLIB.xPCTarget
Return	If the method detects an error, it returns 0. Otherwise, it returns -1.
Description	The <code>xPCTarget.UnLoadApp</code> method stops the current target application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new target application. The method <code>xPCTarget.LoadApp</code> calls this method before loading a new target application.

xPCTarget.UnLoadApp

A

- applications
 - deploying 4-39
 - xPC Target C API 3-1
 - xPC Target COM API 4-1

B

- block parameters
 - tagging 4-8
- block signals
 - tagging 4-12

C

- COM API methods
 - communication objects 8-2
 - file system objects 8-7
 - parameter objects, 8-6
 - scope objects 8-4
 - signal objects, 8-6
 - target objects 8-3
 - xPCFileSystem 8-7
 - xPCProtocol 8-2
 - xPCScopes 8-4 8-6
 - xPCTarget 8-3
- COM applications 4-1
 - before you start 4-2
 - examples 5-1
 - Microsoft Visual Basic .NET example 5-2
 - Microsoft Visual Basic 6.0 example 5-5
 - Tcl/Tk scripts 5-8
 - Visual Basic 4-4
 - xpctank 4-5
- COM Test Harness 8-10 to 8-11 8-14 to 8-56 8-58 to 8-63 8-65 to 8-66 8-68 to 8-80 8-82 to 8-115 8-117 to 8-127 8-129 to 8-139
- custom applications
 - Visual C example 3-4
 - xPC Target COM API 4-1

- custom C applications
 - before you start 3-2
 - example 3-4
 - guidelines 3-2
- custom GUI
 - COM objects 4-4
- custom Visual Basic applications
 - before you start 4-2
 - building 4-39
 - creating 4-19
 - creating event procedures 4-28
 - creating general declarations 4-27
 - creating load procedure 4-27
 - example 4-4
 - graphical interface 4-23
 - setting properties 4-25
 - writing code 4-26

D

- dirStruct structure 7-15
- diskinfo structure 7-17

F

- fileinfo structure 7-20
- FSDir structure 8-10
- FSDiskInfo structure 8-11

G

- GUI creation 4-1

L

- lgmode structure 7-21

M

- model-specific COM library
 - classes 4-17

- creating 4-14
- parameter class 4-18
- referencing 4-21
- signal class 4-18

P

- parameters
 - tagging 4-8

S

- scopedata structure 7-22
- signals
 - tagging 4-12

T

- tagging block parameters 4-8
- tagging block signals 4-12
- target application
 - running API application 3-10
 - using 3-10
- target applications
 - building 3-6

V

- Visual Basic example 4-4
 - building 4-39
 - creating event procedures 4-28
 - creating general declarations 4-27
 - creating load procedure 4-27
 - creating new 4-19
 - graphical interface 4-23
 - referencing without tags 4-34
 - setting properties 4-25
 - tagging block parameters 4-8
 - tagging block signals 4-12
 - testing 4-38
 - writing code 4-26

- Visual C example
 - building 3-9
 - C code 3-16
 - creating 3-6

X

- xPC Target API for Microsoft® .NET Framework
 - overview 1-2
- xPC Target™ API for Microsoft® .NET Framework 2-1
- xPC Target APIs
 - overview 1-2
- xPC Target C API 3-1
 - application creation 3-1
 - example 3-4
 - introduction 1-5
 - overview 1-2
- xPC Target COM API 4-1
 - GUI creation 4-1
 - introduction 1-7
 - overview 1-2
- xPCAddScope function 7-25
- xPCAveragetET function 7-26
- xPCCloseConnection function 7-27
- xPCClosePort function 7-28
- xPCDeRegisterTarget function 7-29
- xPCErrorMsg function 7-30
- xPCFileSystem 8-7
 - xPCFileSystem.CD method 8-14
 - xPCFileSystem.CloseFile method 8-15
 - xPCFileSystem.DirList method 8-16
 - xPCFileSystem.GetDiskInfo method 8-17
 - xPCFileSystem.GetFileSize method 8-18
 - xPCFileSystem.Init method 8-19
 - xPCFileSystem.MKDIR method 8-20
 - xPCFileSystem.OpenFile method 8-21
 - xPCFileSystem.PWD method 8-22
 - xPCFileSystem.ReadFile method 8-23
 - xPCFileSystem.RemoveFile method 8-24

xPCFileSystem.RMDIR method 8-25
xPCFileSystem.ScGetFileName method 8-26
xPCFileSystem.ScGetWriteMode method 8-27
xPCFileSystem.ScGetWriteSize method 8-28
xPCFileSystem.ScSetFileName method 8-29
xPCFileSystem.ScSetWriteMode method 8-30
xPCFileSystem.ScSetWriteSize method 8-31
xPCFileSystem.WriteFile method 8-32
xPCFreeAPI function 7-31
xPCFSCD function 7-32
xPCFSCloseFile function 7-33
xPCFSDir function 7-34
xPCFSDirItems function 7-35
xPCFSDirSize function 7-36
xPCFSDirStructSize function 7-37
xPCFSDiskInfo function 7-38
xPCFSFileInfo function 7-39
xPCFSGetError function 7-40
xPCFSGetFileSize function 7-41
xPCFSGetPWD function 7-42
xPCFSMKDIR function 7-43
xPCFSOpenFile function 7-44
xPCFSReadFile function 7-45
xPCFSRemoveFile function 7-46
xPCFSRMDIR function 7-47
xPCFSScGetFilename function 7-48
xPCFSScGetWriteMode function 7-49
xPCFSScGetWriteSize function 7-50
xPCFSScSetFilename function 7-51
xPCFSScSetWriteMode function 7-52
xPCFSScSetWriteSize function 7-53
xPCFSWriteFile function 7-54
xPCGetAPIVersion function 7-55
xPCGetAppName function 7-56
xPCGetEcho function 7-57
xPCGetExecTime function 7-58
xPCGetLastError function 7-59
xPCGetLoadTimeOut function 7-60
xPCGetLogMode function 7-62
xPCGetNumOutputs function 7-63
xPCGetNumParams function 7-64
xPCGetNumScopes function 7-65
xPCGetNumScSignals function 7-66
xPCGetNumSignals function 7-67
xPCGetNumStates function 7-68
xPCGetOutputLog function 7-69
xPCGetParam function 7-71
xPCGetParamDims function 7-72
xPCGetParamIdx function 7-73
xPCGetParamName function 7-74
xPCGetSampleTime function 7-75
xPCGetScope function 7-76
xPCGetScopeList function 7-77
xPCGetScopes function 7-78
xPCGetSessionTime function 7-79
xPCGetSigIdxfromLabel function 7-82
xPCGetSigLabelWidth function 7-84
xPCGetSignal function 7-80
xPCGetSignalIdx function 7-81
xPCGetSignalLabel function 7-83
xPCGetSignalName function 7-85
xPCGetSignals function 7-86
xPCGetSignalWidth function 7-88
xPCGetStateLog function 7-89
xPCGetStopTime function 7-91
xPCGetTargetVersion function 7-92
xPCGetTETLog function 7-93
xPCGetTimeLog function 7-94
xPCInitAPI function 7-95
xPCIsAppRunning function 7-96
xPCIsOverloaded function 7-97
xPCIsScFinished function 7-98
xPCLoadApp function 7-99
xPCLoadParamSet function 7-101
xPCMaximumTET function 7-103
xPCMaxLogSamples function 7-102
xPCMinimumTET function 7-104
xPCNumLogSamples function 7-105
xPCNumLogWraps function 7-106
xPCOpenConnection function 7-107

xPCOpenSerialPort function 7-108
xPCOpenTcpIpPort function 7-109
xPCParameters 8-6
xPCProtocol 8-2
xPCProtocol.Close method 8-33
xPCProtocol.GetLoadTimeOut method 8-34
xPCProtocol.GetxPCErrorMsg method 8-35
xPCProtocol.Init method 8-36
xPCProtocol.isxPCError method 8-37
xPCProtocol.Port method 8-38
xPCProtocol.RS232Connect method 8-40
xPCProtocol.SetLoadTimeOut method 8-41
xPCProtocol.TargetPing method 8-42
xPCProtocol.TcpIpConnect method 8-43
xPCProtocol.Term method 8-44
xPCProtocol.xPCReboot method 8-39
xPCReboot function 7-110
xPCRegisterTarget function 7-112
xPCRemScope function 7-114
xPCReOpenPort function 7-111
xPCSaveParamSet function 7-115
xPCScAddSignal function 7-116
xPCScGetData function 7-118
xPCScGetDecimation function 7-120
xPCScGetNumPrePostSamples function 7-121
xPCScGetNumSamples function 7-122
xPCScGetNumSignals function 7-123
xPCScGetSignalList function 7-124
xPCScGetSignals function 7-125
xPCScGetStartTime function 7-126
xPCScGetState function 7-127
xPCScGetTriggerLevel function 7-129
xPCScGetTriggerMode function 7-130
xPCScGetTriggerScope function 7-132
xPCScGetTriggerScopeSample function 7-133
xPCScGetTriggerSignal function 7-134
xPCScGetTriggerSlope function 7-135
xPCScGetType function 7-137
xPCScopes 8-4
xPCScopes.AddFileScope method 8-45
xPCScopes.AddHostScope method 8-46
xPCScopes.AddTargetScope method 8-47
xPCScopes.GetScopes method 8-48
xPCScopes.GetxPCError method 8-49
xPCScopes.Init method 8-50
xPCScopes.IsScopeFinished method 8-51
xPCScopes.IsxPCError method 8-52
xPCScopes.RemScope method 8-53
xPCScopes.ScopeAddSignal method 8-54
xPCScopes.ScopeGetAutoRestart method 8-55
xPCScopes.ScopeGetData method 8-56
xPCScopes.ScopeGetDecimation method 8-58
xPCScopes.ScopeGetNumPrePostSamples
method 8-59
xPCScopes.ScopeGetNumSamples method 8-60
xPCScopes.ScopeGetSignals method 8-61
xPCScopes.ScopeGetStartTime method 8-62
xPCScopes.ScopeGetState method 8-63
xPCScopes.ScopeGetTriggerLevel
method 8-65
xPCScopes.ScopeGetTriggerMode method 8-66
xPCScopes.ScopeGetTriggerModeStr
method 8-68
xPCScopes.ScopeGetTriggerScopeSample
method 8-69
xPCScopes.ScopeGetTriggerSignal
method 8-70
xPCScopes.ScopeGetTriggerSlope
method 8-71
xPCScopes.ScopeGetTriggerSlopeStr
method 8-72
xPCScopes.ScopeGetType method 8-73
xPCScopes.ScopeRemSignal method 8-74
xPCScopes.ScopeSetAutoRestart method 8-75
xPCScopes.ScopeSetDecimation method 8-76
xPCScopes.ScopeSetNumPrePostSamples
method 8-77
xPCScopes.ScopeSetNumSamples method 8-78
xPCScopes.ScopeSetTriggerLevel
method 8-79

- xPCScopes.ScopeSetTriggerMode method 8-80
- xPCScopes.ScopeSetTriggerScopeSample method 8-82
- xPCScopes.ScopeSetTriggerSignal method 8-83
- xPCScopes.ScopeSetTriggerSlope method 8-84
- xPCScopes.ScopeSoftwareTrigger method 8-85
- xPCScopes.ScopeStart method 8-86
- xPCScopes.ScopeStop method 8-87
- xPCScopes.TargetScopeGetGrid method 8-88
- xPCScopes.TargetScopeGetMode method 8-89
- xPCScopes.TargetScopeGetModeStr method 8-90
- xPCScopes.TargetScopeGetViewMode method 8-91
- xPCScopes.TargetScopeGetYLimits method 8-92
- xPCScopes.TargetScopeSetGrid method 8-93
- xPCScopes.TargetScopeSetMode method 8-94
- xPCScopes.TargetScopeSetViewMode method 8-95
- xPCScopes.TargetScopeSetYLimits method 8-96
- xPCScRemSignal function 7-138
- xPCScSetDecimation function 7-140
- xPCScSetNumPrePostSamples function 7-141
- xPCScSetNumSamples function 7-142
- xPCScSetTriggerLevel function 7-143
- xPCScSetTriggerMode function 7-144
- xPCScSetTriggerScope function 7-146
- xPCScSetTriggerScopeSample function 7-147
- xPCScSetTriggerSignal function 7-148
- xPCScSetTriggerSlope function 7-149
- xPCScSoftwareTrigger function 7-151
- xPCScStart function 7-152
- xPCScStop function 7-153
- xPCSetEcho function 7-154
- xPCSetLastError function 7-155
- xPCSetLoadTimeOut function 7-156
- xPCSetLogMode function 7-157
- xPCSetParam function 7-158
- xPCSetSampleTime function 7-159
- xPCSetScope function 7-160
- xPCSetStopTime function 7-161
- xPCSignals 8-6
- xPCStartApp function 7-162
- xPCStopApp function 7-163
- xpctank model 4-5
- xPCTarget 8-3
- xPCTarget.AverageTET method 8-97
- xPCTarget.GetAppName method 8-98
- xPCTarget.GetExecTime method 8-99
- xPCTarget.GetNumOutputs method 8-100
- xPCTarget.GetNumParams method 8-101
- xPCTarget.GetNumSignals method 8-102
- xPCTarget.GetNumStates method 8-103
- xPCTarget.GetOutputLog method 8-104
- xPCTarget.GetParam method 8-105
- xPCTarget.GetParamDims method 8-106
- xPCTarget.GetParamIdx method 8-107
- xPCTarget.GetParamName method 8-108
- xPCTarget.GetSampleTime method 8-109
- xPCTarget.GetSignal method 8-110
- xPCTarget.GetSignalIdsfromLabel method 8-111
- xPCTarget.GetSignalIdx method 8-113
- xPCTarget.GetSignalLabel method 8-112
- xPCTarget.GetSignalName method 8-114
- xPCTarget.GetSignals method 8-115
- xPCTarget.GetSignalWidth method 8-117
- xPCTarget.GetStateLog method 8-118
- xPCTarget.GetStopTime method 8-119
- xPCTarget.GetTETLog method 8-120
- xPCTarget.GetTimeLog method 8-121
- xPCTarget.GetxPCError method 8-122
- xPCTarget.Init method 8-123
- xPCTarget.IsAppRunning method 8-124
- xPCTarget.IsOverloaded method 8-125

xPCTarget.isPCError method 8-126
xPCTarget.LoadApp method 8-127
xPCTarget.MaximumTET method 8-129
xPCTarget.MaxLogSamples method 8-130
xPCTarget.MinimumTET method 8-131
xPCTarget.NumLogSamples method 8-132
xPCTarget.NumLogWraps method 8-133
xPCTarget.SetParam method 8-134
xPCTarget.SetSampleTime method 8-135
xPCTarget.SetStopTime method 8-136
xPCTarget.StartApp method 8-137
xPCTarget.StopApp method 8-138
xPCTarget.UnloadApp method 8-139
xPCTargetPing function 7-164
xPCTgScGetGrid function 7-165
xPCTgScGetMode function 7-166
xPCTgScGetViewMode function 7-167
xPCTgScGetYLimits function 7-168
xPCTgScSetGrid function 7-169
xPCTgScSetMode function 7-170
xPCTgScSetViewMode function 7-171
xPCTgScSetYLimits function 7-172
xPCUnloadApp function 7-173