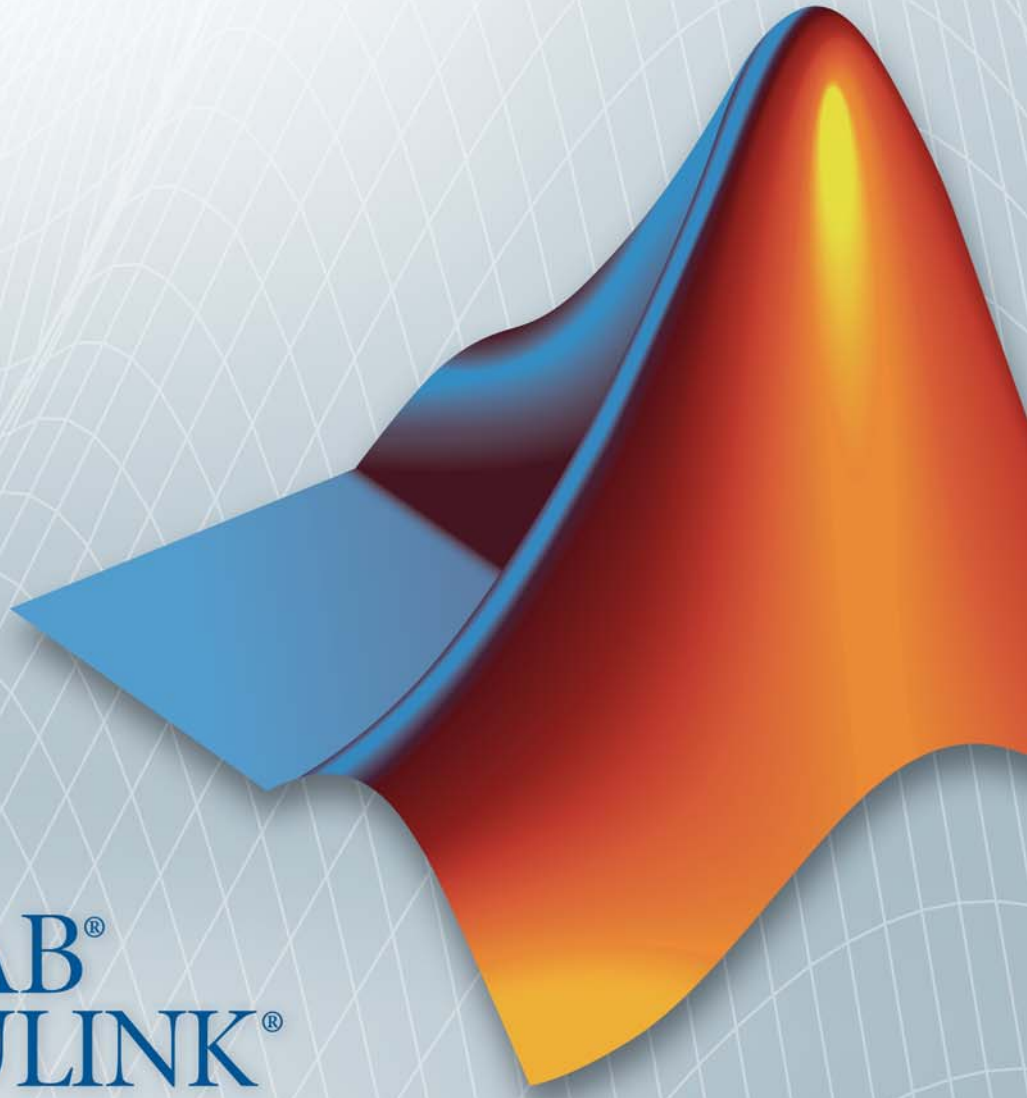
xPC Target™

Reference

R2012b



MATLAB®
&SIMULINK®

MathWorks®

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

xPC Target™ Reference

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

March 2007	Online only	New for Version 3.2 (Release 2007a)
September 2007	Online only	Updated for Version 3.3 (Release 2007b)
March 2008	Online only	Updated for Version 3.4 (Release 2008a)
October 2008	Online only	Updated for Version 4.0 (Release 2008b)
March 2009	Online only	Updated for Version 4.1 (Release 2009a)
September 2009	Online only	Updated for Version 4.2 (Release 2009b)
March 2010	Online only	Updated for Version 4.3 (Release 2010a)
April 2011	Online only	Updated for Version 5.0 (Release 2011a)
September 2011	Online only	Updated 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)

Contents

1 Function Reference

2 Functions

3 xPC Target API Reference for C

4 xPC Target API Reference for COM

5 Configuration Parameters

6 Target Computer Command-Line Interface Reference

1

Function Reference

Classes

`xpctarget Package`	Package for all xPC Target MATLAB classes
`xpctarget.env Class`	Stores target environment properties
`xpctarget.fs Class`	Manage the directories and files on the target computer
`xpctarget.fsbase Class`	Base class of file system and file transfer protocol (FTP) classes
`xpctarget.ftp Class`	Manage the directories and files on the target computer via file transfer protocol (FTP)
`xpctarget.targets Class`	Container object to manage target computer environment collection objects
`xpctarget.xpc Class`	Target object representing target application
`xpctarget.xpcfs Class`	Control and access properties of file scopes
`xpctarget.xpcsc Class`	Base class for all scope classes
`xpctarget.xpcschost Class`	Control and access properties of host scopes
`xpctarget.xpcsctg Class`	Control and access properties of target scopes

Target Computers

`getxpcinfo`	Retrieve diagnostic information to help troubleshoot configuration issues
`macaddr`	Convert string-based MAC address to vector-based one
`xpcbench`	benchmark
`xpcbootdisk`	Create xPC Target boot disk or DOS Loader files and confirm current environment properties
`xpcbytes2file`	Generate file suitable for use by From File block
`xpcexplr`	Open xPC Target Explorer
`xpcgetCC`	Compiler settings for xPC Target environment
`xpcnetboot`	Create kernel to boot target computer over dedicated network
`xpcsetCC`	Compiler settings for xPC Target environment
`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.getxpcpci`	Determine which PCI boards are installed in target computer
`xpctarget.xpc.targetping`	Test communication between host and target computers
`xpctargetping`	Test communication between host and target computers
`xpctargetspy`	Open Real-Time xPC Target Spy window on host computer
`xpctest`	Test xPC Target installation
`xpcwwwenable`	Disconnect target computer from current client application

Target Environments

`getxpcenv`	List environment properties assigned to MATLAB® variable
`setxpcenv`	Change xPC Target environment properties
`xpctarget.env.get (env object)`	Return target environment property values
`xpctarget.env.set (env object)`	Change target environment object property values
`xpctarget.targets`	Create container object to manage target computer environment collection objects
`xpctarget.targets.Add (env collection object)`	Add new xPC Target environment collection object
`xpctarget.targets.get (env collection object)`	Return target object collection environment property values
`xpctarget.targets.getTargetNames (env collection object)`	Retrieve xPC Target environment object names
`xpctarget.targets.Item (env collection object)`	Retrieve specific xPC Target environment (env) object
`xpctarget.targets.makeDefault (env collection object)`	Set specific target computer environment object as default
`xpctarget.targets.Remove (env collection object)`	Remove specific xPC Target environment object
`xpctarget.targets.set (env collection object)`	Change target object environment collection object property values

Target Applications

`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.close`	Close serial port connecting host computer with target computer
`xpctarget.xpc.get (target application object)`	Return target application object property values
`xpctarget.xpc.load`	Download target application to target computer
`xpctarget.xpc.reboot`	Reboot target computer
`xpctarget.xpc.set (target application object)`	Change target application object property values
`xpctarget.xpc.start (target application object)`	Start execution of target application on target computer
`xpctarget.xpc.stop (target application object)`	Stop execution of target application on target computer
`xpctarget.xpc.unload`	Remove current target application from target computer

Scopes

`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.addscope`	Create scopes
`xpctarget.xpc.getscope`	Scope object pointing to scope defined in kernel
`xpctarget.xpc.remscope`	Remove scope from target computer
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get (scope object)`	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set (scope object)`	Change property values for scope objects
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software-trigger start of data acquisition for scope(s)

Parameters

`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.getparam`	Value of target object parameter index
`xpctarget.xpc.getparamid`	Parameter index from parameter list
`xpctarget.xpc.getparamname`	Block path and parameter name from index list
`xpctarget.xpc.loadparamset`	Restore parameter values saved in specified file
`xpctarget.xpc.saveparamset`	Save current target application parameter values
`xpctarget.xpc.setparam`	Change writable target object parameters

Signals

`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.getsignal`	Value of target object signal index
`xpctarget.xpc.getsignalid`	Signal index or signal property from signal list
`xpctarget.xpc.getsignalidsfromlabel`	Return vector of signal indices
`xpctarget.xpc.getsignallabel`	Return signal label
`xpctarget.xpc.getsignalname`	Signal name from index list

Data Logs

`xpctarget.xpc`	Create target object representing target application
`xpctarget.xpc.getlog`	All or part of output logs from target object

File Systems

`xpctarget.fs`	Create xPC Target file system object
`xpctarget.fs.diskinfo`	Information about target computer drive
`xpctarget.fs.fclose`	Close open target computer file(s)
`xpctarget.fs.fileinfo`	Target computer file information
`xpctarget.fs.filetable`	Information about open files in target computer file system
`xpctarget.fs.fopen`	Open target computer file for reading
`xpctarget.fs.fread`	Read open target computer file
`xpctarget.fs.fwrite`	Write binary data to open target computer file
`xpctarget.fs.getfilesize`	Size of file on target computer
`xpctarget.fs.readxpcfile`	Interpret raw data from xPC Target file format
`xpctarget.fs.removefile`	Remove file from target computer
`xpctarget.fs.selectdrive`	Select target computer drive
`xpctarget.fsbase.cd`	Change folder on target computer
`xpctarget.fsbase.dir`	List contents of current folder on target computer
`xpctarget.fsbase.mkdir`	Make folder on target computer
`xpctarget.fsbase.pwd`	Current folder path of target computer
`xpctarget.fsbase.rmdir`	Remove folder from target computer
`xpctarget.ftp`	Create file transfer protocol (FTP) object
`xpctarget.ftp.get (ftp)`	Retrieve copy of requested file from target computer
`xpctarget.ftp.put`	Copy file from host computer to target computer

2

Functions

fc422mexcalcbits

Purpose

Calculate parameter values for Fastcom 422/2-PCI board

Syntax

MATLAB command line

```
[a b ] = fc422mexcalcbits(frequency)
[a b df] = fc422mexcalcbits(frequency)
```

Arguments

`frequency` Desired baud rate for the board

`[a b] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver clock. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

`[a b df] = fc422mexcalcbits(frequency)` accepts a baud rate (in units of baud/second) and converts this value into two parameters `a b`. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver block. The third value, `df`, indicates the actual baud rate that is created by the generated parameters `a b`. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency. The desired baud rate (`frequency`) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

Purpose

List environment properties assigned to MATLAB variable

Syntax

MATLAB command line

```
getxpcenv
```

Description

Function to list environment properties. This function displays, in the MATLAB Command Window, the property names, the current property values, and the new property values set for the xPC Target environment.

The environment properties define communication between the host computer and target computer and the type of target boot kernel created during the setup process.

Tip To access a subset of these properties in xPC Target Explorer:

1. Expand a target computer node in the **Targets** pane.
2. Click the Target Properties icon  in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 2-4
- "Target Settings" on page 2-10
- "Boot Configuration" on page 2-14
- "Host Configuration" on page 2-16

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer. If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`. **Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. Select 1200, 2400, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.

Environment Property	Description
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
`TcpIpTargetBusType`	Property values are `'PCI'`, `'ISA'`, and `` `USB' ``. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.

Environment Property	Description
	If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.
`TcpIpTargetDriver`	Property values are `'3C90x'`, `` `I8254x' ``, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `` `Rhine' ``, `'RTLANCE'`, `'SMC91C9X'`, `` `USBAX772' ``, `` `USBAX172' ``, and `` `Auto' ``. Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.

Environment Property	Description
`TcpIpTargetISAIRQ`	Property value is `'n'`, where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.

Environment Property	Description
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`. Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`. Enter a port address greater than `20000` in the **Port** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `22222`. The default value is higher than the reserved area

Environment Property	Description
	(`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` and `'off'` (the default). Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.
`MaxModelSize`	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`. Select `1 MB`, `4 MB`, or `16 MB` from the **Model size** list in the **Target Properties** pane of xPC Target Explorer. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve

Environment Property	Description
	enough memory for the target application and creates an error. **Note** • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.

Environment Property	Description
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `` `on' `` and `` `off' `` (the default). If you create a target boot kernel when `ShowHardware` is `` `on' `` and boot the target computer with it, the kernel displays on the target monitor the index, bus, slot, function, and target driver for each Ethernet card. **Note** The host computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'Manual'`. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If `TargetRAMSizeMB` is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not

Environment Property	Description
	contain more than 64 MB of RAM or you do not want to use more than 64 MB, select `Auto`. If the target computer has more than 64 MB of RAM and you want to use more than 64 MB, select `Manual` and enter the amount of RAM installed in the target computer. **Note** The xPC Target kernel can use only 2 GB of memory.
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. **Tip** To use all the features of the target scope, you also need to install a keyboard on the target computer.
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`CDBootImageLocation`	Location of `cdboot.iso` file for creation of CD target boot disk.
`DOSLoaderLocation`	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
`EmbeddedOption`	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option™ product is enabled only if you purchase an additional license.
`TargetBoot`	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer. If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded

Environment Property	Description
	Option product licensed and installed, you have the additional choice of `Stand Alone`. **Tip** Click the **Create boot disk** button to create a bootable image in the specified boot mode.
`TargetMACAddress`	Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Host Configuration

Environment Property	Description
`Version`	xPC Target version number. Displayed only from `getxpcenv` when called without arguments.

Examples

Return the xPC Target environment in the structure shown below. The output in the MATLAB window is suppressed. The structure contains three fields for property names, current property values, and new property values.

```
env = getxpcenv
env =
      propname: {1x25 cell}
    actpropval: {1x25 cell}
    newpropval: {1x25 cell}
```

Display a list of the environment property names, current values, and new values.

```
env = getxpcenv
```

See Also

`setxpcenv` | `xpcbootdisk`

Purpose

Retrieve diagnostic information to help troubleshoot configuration issues

Syntax

MATLAB command line

```
getxpcinfo
getxpcinfo('-a')
```

Arguments

`'-a'`	Appends diagnostic information to an existing `xpcinfo.txt` file. If one does not exist, this function creates the file in the current folder.

Description

`getxpcinfo` returns diagnostic information for troubleshooting xPC Target configuration issues. This function generates and saves the information in the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` already exists, this function overwrites it with the new information.

`getxpcinfo('-a')` appends the diagnostic information to the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` does not exist, this function creates it.

You can send the file `xpcinfo.txt` to MathWorks Technical Support for evaluation and guidance. To create this file, you must have write permission for the current folder.

Warning

The file `xpcinfo.txt` might contain information sensitive to your organization. Review the contents of this file before sending to MathWorks.

macaddr

Purpose

Convert string-based MAC address to vector-based one

Syntax

MATLAB command line

```
macaddr('MAC address')
```

Argument

`'MAC address'`	String-based MAC address to be converted.

Description

The `macaddr` function converts a string-based MAC address to a vector-based MAC address. The string-based MAC address should be a string comprised of six colon-delimited fields of two-digit hexadecimal numbers.

Examples

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

ans =

    1   35   69  103  137  171
```

How To

- "Model-Based Ethernet Communications"

Purpose

Change xPC Target environment properties

Syntax

MATLAB command line

```
setxpcenv('property_name', 'property_value')
setxpcenv('prop_name1', 'prop_val1', 'prop_name2',
'prop_val2')
setxpcenv
```

Arguments

`property_name`	Not case sensitive. Property names can be shortened as long as they can be differentiated from the other property names.
`property_value`	Character string. Type `setxpcenv` without arguments to get a listing of allowed values. Property values are not case sensitive.

Description

Function to enter new values for environment properties. If the new value is different from the current value, the property is marked as having a new value.

The environment properties define communication between the host computer and target computer and the type of target boot kernel created during the setup process. With the exception of the `Version` property, you can set environment properties using the `setxpcenv` function or the xPC Target Explorer window, accessed via the `xpcexplr` function. An understanding of the environment properties will help you configure the xPC Target environment.

The function `setxpcenv` works similarly to the `set` function of the MATLAB Handle Graphics® system. Call the function `setxpcenv` with an even number of arguments. The first argument of a pair is the property name; the second argument is the new property value for this property.

Using the function `setxpcenv` without arguments returns a list of allowed property values in the MATLAB window.

Tip To access a subset of these properties in xPC Target Explorer:

1. Expand a target computer node in the **Targets** pane.
2. Click the Target Properties icon  in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 2-20
- "Target Settings" on page 2-26
- "Boot Configuration" on page 2-30

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer. If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. Select `1200`, `2400`, `48 00`, `9600`, `19200`, `38400`, `57600`, or `115200` from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
`TcpIpTargetBusType`	Property values are `'PCI'`, `'ISA'`, and `` `USB' ``. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
`TcpIpTargetDriver`	Property values are `'3C90x'`, `` `I8254x' ``, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `` `Rhine' ``, `'RTLANCE'`, `'SMC91C9X'`, `` `USBAX772' ``, `` `USBAX172' ``, and `` `Auto' ``. Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
`TcpIpTargetISAIRQ`	Property value is `'n'`, where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`. Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`. Enter a port address greater than `20000` in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer. This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` and `'off'` (the default). Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.

Environment Property	Description
`MaxModelSize`	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`. Select `1 MB`, `4 MB`, or `16 MB` from the **Model size** list in the **Target Properties** pane of xPC Target Explorer. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. **Note** • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.

Environment Property	Description
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `` `on' `` and `` `off' `` (the default). If you create a target boot kernel when `ShowHardware` is `` `on' `` and boot the target computer with it, the kernel displays on the target monitor the index, bus, slot, function, and target driver for each Ethernet card. **Note** The host computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'Manual'`. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If `TargetRAMSizeMB` is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM or you do not want to use more than 64 MB, select `Auto`. If the target computer has more than 64 MB of RAM and you want to use more than 64 MB, select `Manual` and enter the amount of RAM installed in the target computer. **Note** The xPC Target kernel can use only 2 GB of memory.

Environment Property	Description
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. **Tip** To use all the features of the target scope, you also need to install a keyboard on the target computer.
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`CDBootImageLocation`	Location of `cdboot.iso` file for creation of CD target boot disk.
`DOSLoaderLocation`	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
`EmbeddedOption`	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.
`TargetBoot`	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer. If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of `Stand Alone`. **Tip** Click the **Create boot disk** button to create a bootable image in the specified boot mode.
`TargetMACAddress`	Physical target computer MAC address from which to accept boot

Environment Property	Description
	requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Examples

List the current environment properties.

```
setxpcenv
```

Change the serial communication port of the host computer to `COM2`.

```
setxpcenv('RS232HostPort','COM2')
```

See Also

`getxpcenv | xpcbootdisk`

How To

- "Network Communication Setup"
- "Serial Communication Setup"

- “Target Boot Methods”
- “Command Line Setup for Single Target Computer Systems”
- “Command Line Setup for Multiple Target Computer Systems”

Purpose

xPC Target benchmark

Syntax

MATLAB command line

```
xpcbench(model)
xpcbench(model,P1)
xpcbench(model,P1,P2)
xpcbench(model,P1,P2,P3)
```

Arguments

Argument	Value	Description
`model`	`'minimal'`	Benchmark the default minimal model.
	`'f14'`	Benchmark the default f14 model (one f14 system).
	`'f14*5'`	Benchmark the default f14*5 model (five copies of the f14 system).
	`'f14*10'`	Benchmark the default f14*10 model (ten copies of the f14 system).
	`'f14*25'`	Benchmark the default f14*25 model (25 copies of the f14 system).
	`'this'`	Benchmark all five default models (the default minimal model plus all four default f14 models).

Argument	Value	Description
	`'usermdl'`	Benchmark your model, *usermdl*.
Up to three optional arguments: `P1`, `P2`, and `P3`	`'-reboot'`	Reboot the target computer after testing each model.
	`'-cleanup'`	Delete build files after running benchmarks.
	`'-verbose'`	Plot and display results in the MATLAB Command Window.

Description

`xpcbench` benchmarks the real-time execution performance of xPC Target applications on your target computer and compares the result with prestored benchmark results from other computers.

The prestored target computer benchmark results of five models (applications) are provided, each model compiled using a sampling of the currently supported compilers (see `http://www.mathworks.com/support/compilers/current_release/`). Compare these results with those for your target computer. Results are labeled by CPU type, CPU clock rate, and the compiler used to compile the application.

The five benchmark models are:

Benchmark	Description
minimal	Based on a minimal model consisting of just three blocks (Constant, Gain, Termination). This model has neither continuous nor discrete states. The result of this benchmark gives an impression about the target computer interrupt latencies.
f14	Based on the standard Simulink example model f14. Type `f14` in the MATLAB Command Window to open the model and view the model (62 blocks, 10 continuous states).
f14*5	Five f14 systems modeled in subsystems (310 blocks, 50 continuous states). This benchmark is five times more demanding than benchmark F14.
f14*10	Ten f14 systems (620 blocks, 100 continuous states).
f14*25	25 f14 systems (1550 blocks, 250 continuous states).

`xpcbench` without an argument displays two plot figures, each containing different representations of the prestored target computer benchmark results. The first plot lists, for each target computer tested, the smallest achievable sample time for the five benchmarks, in microseconds. The second plot contains a bar graph of all computers, ranked by relative performance.

Note

- The prestored benchmark results were collected with **Multicore CPU support** disabled. This allows for direct comparison with previous release results.
- A sample time is achievable if any smaller sample time causes CPU overload.

`xpcbench(model)` benchmarks your target computer using argument `model`. You can specify:

- All five default benchmarks (`'this'`)
- One of the five default benchmarks (`'minimal'`, `'f14'`, `'f14*5'`, `'f14*10'`, `'f14*25'`)
- Your model (`'usermdl'`)

Before you run `xpcbench`, you must connect the host machine to the target computer and run the xPC Target test, `xpctest`, with no failures.

Benchmark execution can take several minutes to complete, including:

1. Generating the benchmark models
2. Building and downloading the xPC Target applications
3. Searching for the smallest achievable sample time
4. Displaying the performance results in the MATLAB Command Window, along with the prestored results for the other target computers

`res = xpcbench` returns the prestored benchmark results in a structure array with fields:

Field Name	Contents
Machine	Target computer information string containing CPU type, CPU speed, compiler
BenchResults	Target computer benchmark performance for the five default models 'minimal', 'f14', 'f14*5', 'f14*10', 'f14*25'
Desc	Target computer descriptor string containing machine type, RAM size, cache size

res = xpcbench(model) returns the benchmark results for model in a structure with fields:

Field Name	Contents
Name	Name of model
nBlocks	Number of blocks in model
BuildTime	Elapsed time in seconds to build model
BenchTime	Elapsed time in seconds to run benchmark for model
Tsmin	Minimal achievable sample time in seconds for model

Examples

xpcbench

Prestored benchmark results showing what to expect of representative processors.

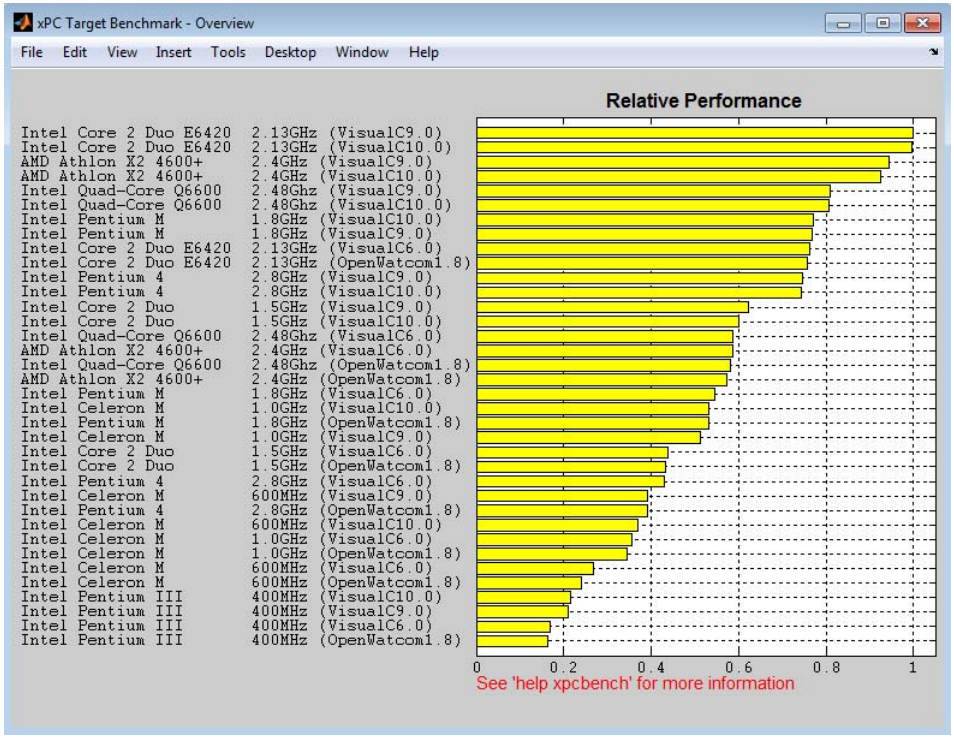
Boot the target computer using your chosen method.

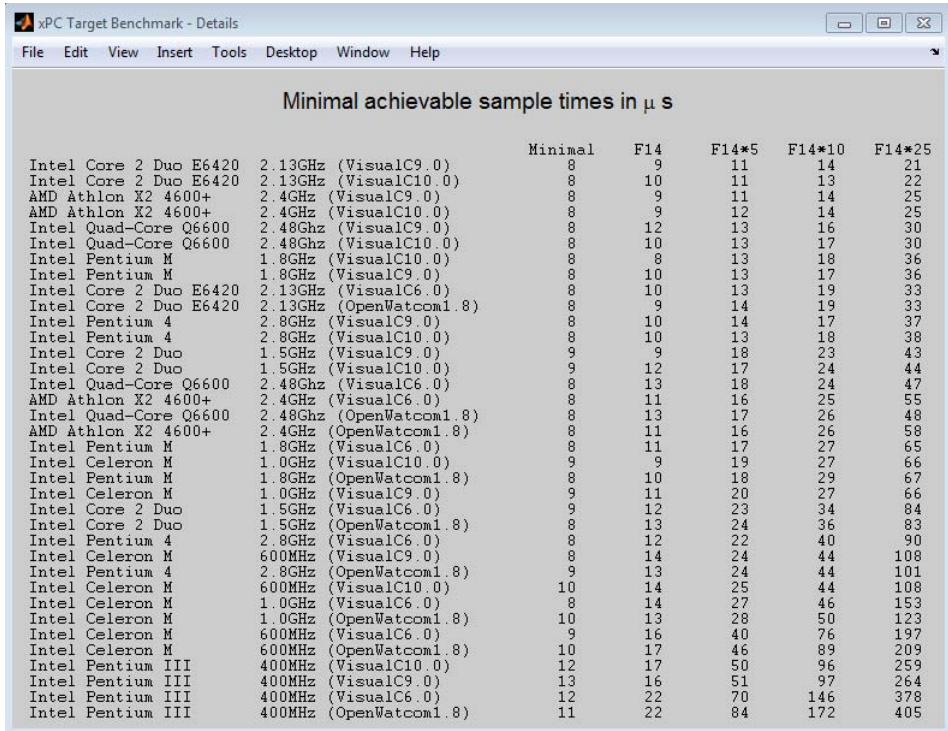
Connect it to the host computer.

```
xpctest

res = xpcbench;
res(1)

ans =

         Machine: 'Intel Celeron M 600MHz (VisualC10.0)'
    BenchResults: [1.0000e-05 1.4000e-05 2.5000e-05
                   4.4000e-05 1.0800e-04]
            Desc: [1x70 char]
```

The array contains benchmark results for each processor type, in arbitrary order.

```
xpcbench
```


xPC Target Benchmark - Overview
File Edit View Insert Tools Desktop Window Help
Relative Performance
Intel Core 2 Duo E6420 2.13GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC10.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC9.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC9.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC6.0)
Intel Core 2 Duo E6420 2.13GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC9.0)
Intel Pentium 4 2.8GHz (VisualC10.0)
Intel Core 2 Duo 1.5GHz (VisualC9.0)
Intel Core 2 Duo 1.5GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC6.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC6.0)
Intel Quad-Core Q6600 2.48Ghz (OpenWatcom1.8)
AMD Athlon X2 4600+ 2.4GHz (OpenWatcom1.8)
Intel Pentium M 1.8GHz (VisualC6.0)
Intel Celeron M 1.0GHz (VisualC10.0)
Intel Pentium M 1.8GHz (OpenWatcom1.8)
Intel Celeron M 1.0GHz (VisualC9.0)
Intel Core 2 Duo 1.5GHz (VisualC6.0)
Intel Core 2 Duo 1.5GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC6.0)
Intel Celeron M 600MHz (VisualC9.0)
Intel Pentium 4 2.8GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC10.0)
Intel Celeron M 1.0GHz (VisualC6.0)
Intel Celeron M 1.0GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC6.0)
Intel Celeron M 600MHz (OpenWatcom1.8)
Intel Pentium III 400MHz (VisualC10.0)
Intel Pentium III 400MHz (VisualC9.0)
Intel Pentium III 400MHz (VisualC6.0)
Intel Pentium III 400MHz (OpenWatcom1.8)
0 0.2 0.4 0.6 0.8 1
See 'help xpcbench' for more information



		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

xpcbench('this')

Benchmark using the five default models.

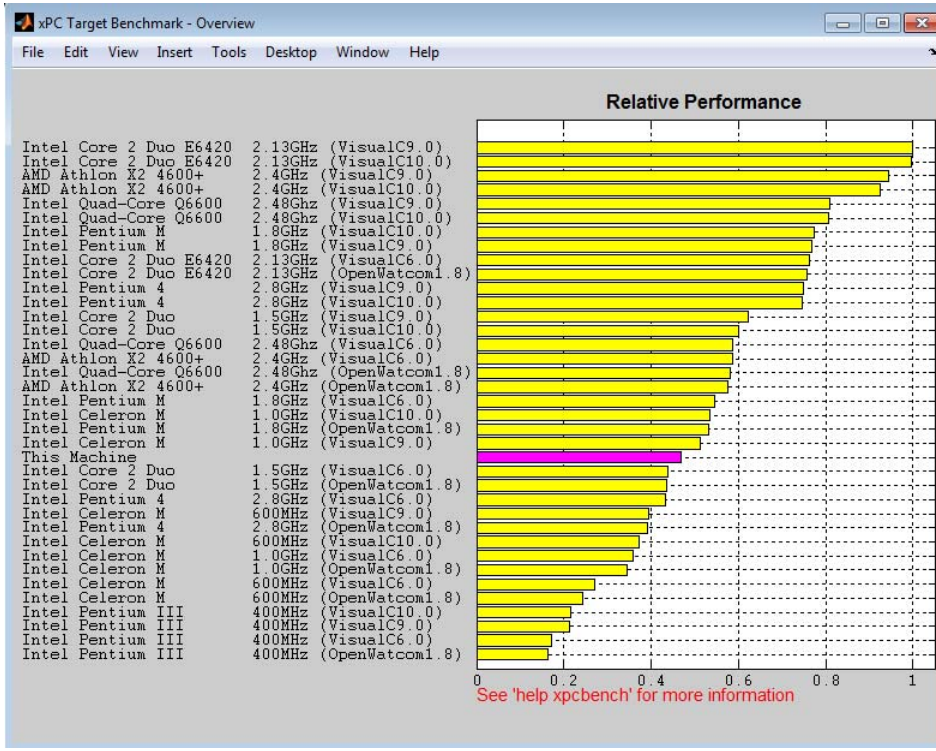
Boot the target computer using your chosen method.

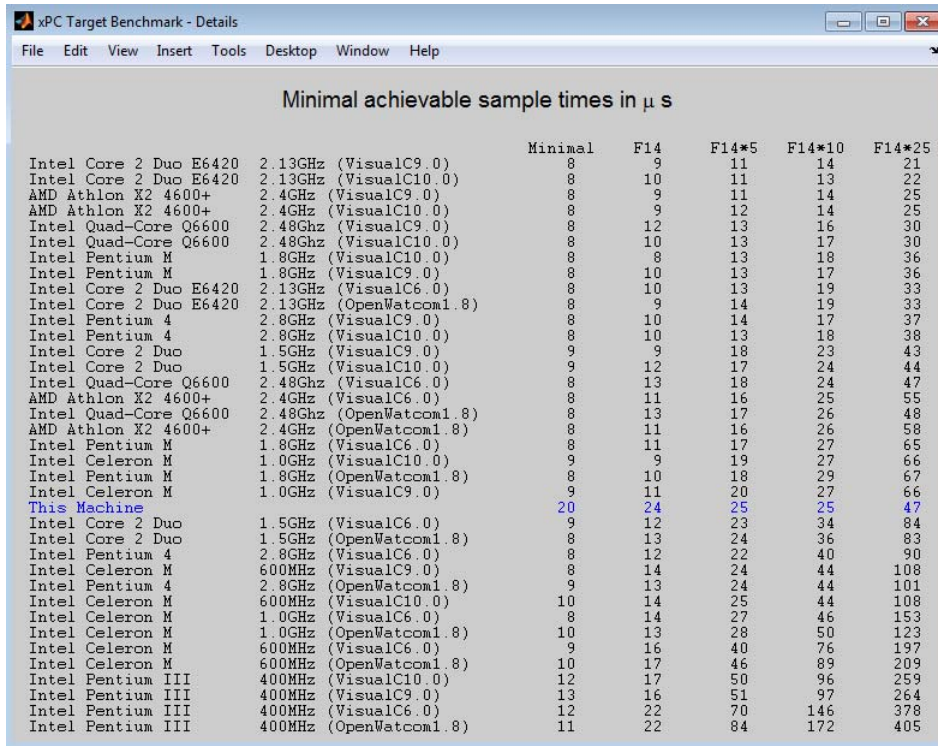
Connect it to the host computer.

```
xpctest

res = xpcbench('this');
res(1)

ans =
```

```
                Name: 'Minimal'
              nBlocks: 3
           BuildTime: 13.0557
           BenchTime: 23.3724
               Tsmin: 1.8656e-05
```

```
xpcbench('this')
```





		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
This Machine		20	24	25	25	47
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

xpcbench('f14*5')

Benchmark using model f14*5 only.

Boot the target computer using your chosen method.

Connect it to the host computer.

```
xpctest
```

```
res = xpcbench('f14*5')
res
```

```
res =
```

```
         Name: 'F14*5'
       nBlocks: 310
    BuildTime: 19.4939
    BenchTime: 29.7564
        Tsmin: 2.4453e-05
```

```
xpcbench('f14*5')
```

```
Benchmark results for model:               F14*5
Number of blocks in model:                 310
Elapsed time for model build (sec):        12.2
Elapsed time for model benchmark (sec):    29.8
Minimal achievable sample time (microsec): 26.3
```

xpcbench('this','-reboot')

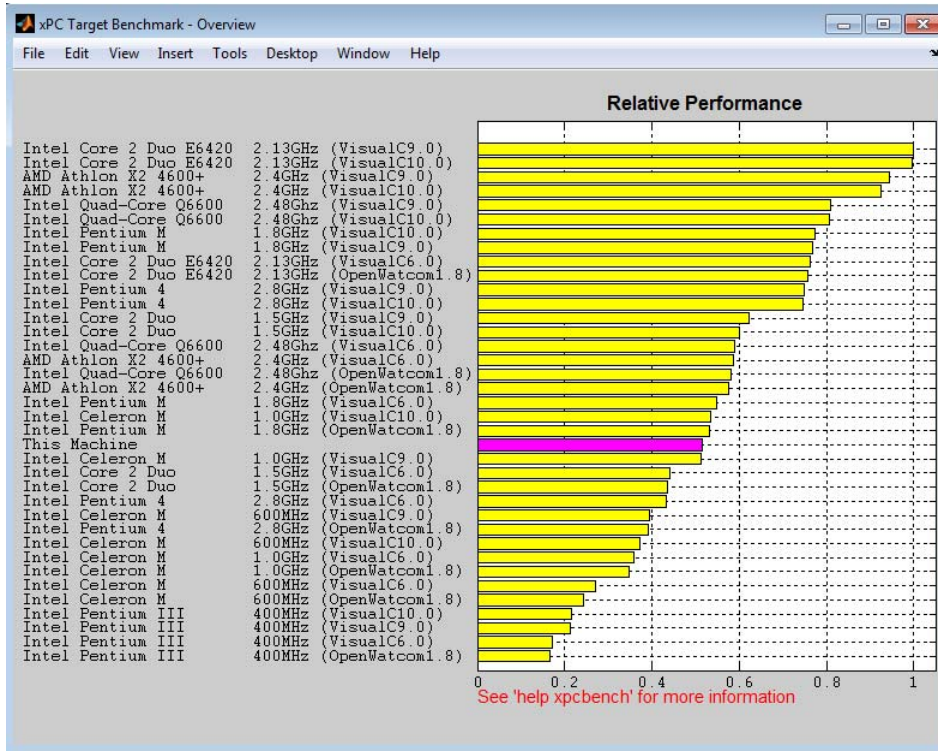
Benchmark the target computer using the five default models, rebooting after each test.

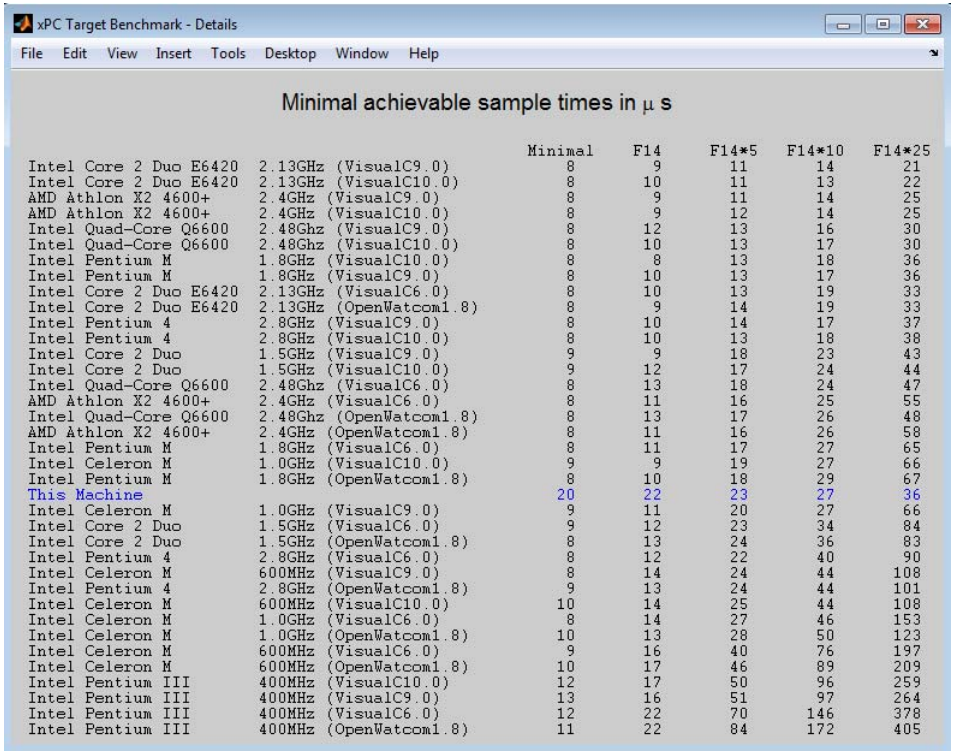
Boot the target computer using your chosen method.

Connect it to the host computer.

```
xpctest
```

```
xpcbench('this','-reboot')
```


xPC Target Benchmark - Overview
File Edit View Insert Tools Desktop Window Help
Relative Performance
Intel Core 2 Duo E6420 2.13GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC10.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC9.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC9.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC6.0)
Intel Core 2 Duo E6420 2.13GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC9.0)
Intel Pentium 4 2.8GHz (VisualC10.0)
Intel Core 2 Duo 1.5GHz (VisualC9.0)
Intel Core 2 Duo 1.5GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC6.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC6.0)
Intel Quad-Core Q6600 2.48Ghz (OpenWatcom1.8)
AMD Athlon X2 4600+ 2.4GHz (OpenWatcom1.8)
Intel Pentium M 1.8GHz (VisualC6.0)
Intel Celeron M 1.0GHz (VisualC10.0)
Intel Pentium M 1.8GHz (OpenWatcom1.8)
This Machine
Intel Celeron M 1.0GHz (VisualC9.0)
Intel Core 2 Duo 1.5GHz (VisualC6.0)
Intel Core 2 Duo 1.5GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC6.0)
Intel Celeron M 600MHz (VisualC9.0)
Intel Pentium 4 2.8GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC10.0)
Intel Celeron M 1.0GHz (VisualC6.0)
Intel Celeron M 1.0GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC6.0)
Intel Celeron M 600MHz (OpenWatcom1.8)
Intel Pentium III 400MHz (VisualC10.0)
Intel Pentium III 400MHz (VisualC9.0)
Intel Pentium III 400MHz (VisualC6.0)
Intel Pentium III 400MHz (OpenWatcom1.8)
0 0.2 0.4 0.6 0.8 1
See 'help xpcbench' for more information



		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
This Machine		20	22	23	27	36
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

xpcbench('xpcosc','-cleanup','-verbose');

Benchmark the target computer using model xpcosc, delete the build files when done, and display full results in the MATLAB Command Window.

Boot the target computer using your chosen method.

Connect it to the host computer.

```
xpctest
```

```
xpcbench('xpcosc','-cleanup','-verbose');
```

```
Benchmark results for model:               xpcosc
Number of blocks in model:                 10
Elapsed time for model build (sec):        37.2
Elapsed time for model benchmark (sec):    23.4
Minimal achievable sample time (microsec): 24.0
```

res = xpcbench('this','-reboot','-cleanup','-verbose');

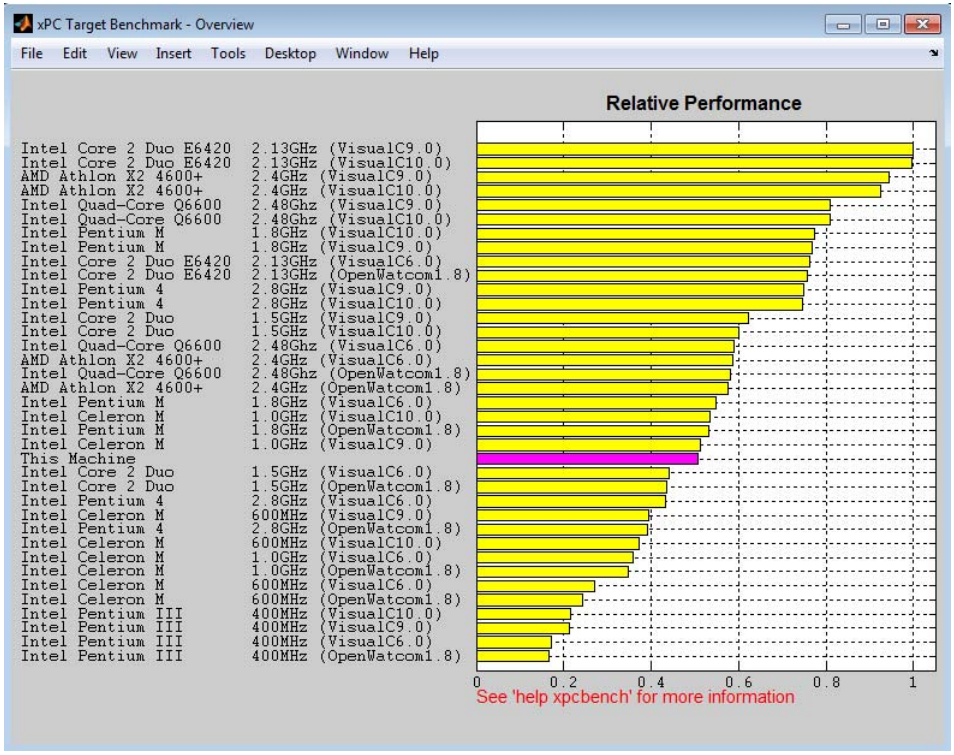
Benchmark the target computer using the five default models, reboot after each test, store the results in *res*, delete the build files when done, and display full results in the MATLAB Command Window.

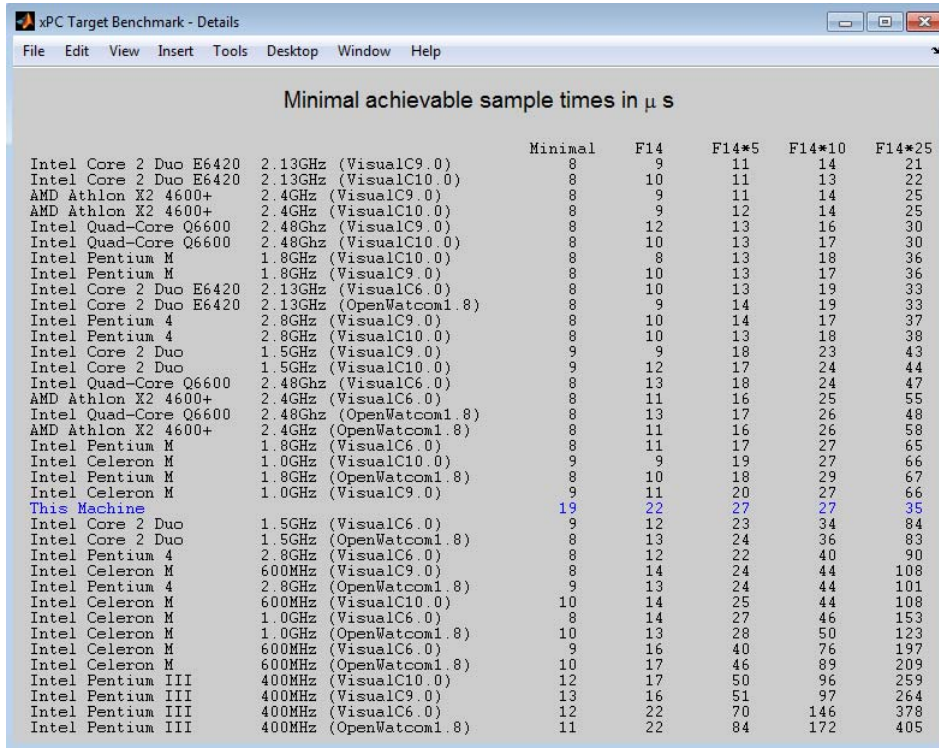
Boot the target computer using your chosen method.

Connect it to the host computer.

```
xpctest
```

```
res = xpcbench('this','-reboot','-cleanup','-verbose');
```


xPC Target Benchmark - Overview
File Edit View Insert Tools Desktop Window Help
Relative Performance
Intel Core 2 Duo E6420 2.13GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC10.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC9.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC9.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC10.0)
Intel Pentium M 1.8GHz (VisualC9.0)
Intel Core 2 Duo E6420 2.13GHz (VisualC6.0)
Intel Core 2 Duo E6420 2.13GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC9.0)
Intel Pentium 4 2.8GHz (VisualC10.0)
Intel Core 2 Duo 1.5GHz (VisualC9.0)
Intel Core 2 Duo 1.5GHz (VisualC10.0)
Intel Quad-Core Q6600 2.48Ghz (VisualC6.0)
AMD Athlon X2 4600+ 2.4GHz (VisualC6.0)
Intel Quad-Core Q6600 2.48Ghz (OpenWatcom1.8)
AMD Athlon X2 4600+ 2.4GHz (OpenWatcom1.8)
Intel Pentium M 1.8GHz (VisualC6.0)
Intel Celeron M 1.0GHz (VisualC10.0)
Intel Pentium M 1.8GHz (OpenWatcom1.8)
Intel Celeron M 1.0GHz (VisualC9.0)
This Machine
Intel Core 2 Duo 1.5GHz (VisualC6.0)
Intel Core 2 Duo 1.5GHz (OpenWatcom1.8)
Intel Pentium 4 2.8GHz (VisualC6.0)
Intel Celeron M 600MHz (VisualC9.0)
Intel Pentium 4 2.8GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC10.0)
Intel Celeron M 1.0GHz (VisualC6.0)
Intel Celeron M 1.0GHz (OpenWatcom1.8)
Intel Celeron M 600MHz (VisualC6.0)
Intel Celeron M 600MHz (OpenWatcom1.8)
Intel Pentium III 400MHz (VisualC10.0)
Intel Pentium III 400MHz (VisualC9.0)
Intel Pentium III 400MHz (VisualC6.0)
Intel Pentium III 400MHz (OpenWatcom1.8)
0 0.2 0.4 0.6 0.8 1
See 'help xpcbench' for more information



		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
This Machine		19	22	27	27	35
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

```
res(1)

ans =

         Name: 'Minimal'
      nBlocks: 3
    BuildTime: 11.0002
    BenchTime: 23.4068
         Tsmin: 1.9141e-05
```

See Also

xpctest

Purpose

Create xPC Target boot disk or DOS Loader files and confirm current environment properties

Syntax

MATLAB command line

```
xpcbootdisk
```

Description

Function to create an xPC Target boot floppy, CD or DVD boot image, network boot image, or DOS Loader files for the current xPC Target environment. Use the `setxpcenv` function to set environment properties.

- Creating an xPC Target boot floppy consists of writing the bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the drive. At the end, a summary of the creation process is displayed.
- Creating an xPC Target CD/DVD boot image consists of creating the bootable kernel image in a designated area. You can then burn the files to a blank CD/DVD. If you have Microsoft® Windows Vista™ or Microsoft Windows® XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), `xpcbootdisk` offers to create to the CD or DVD. Otherwise, you must use alternate third-party CD/DVD writing software to write ISO image files.
- Creating an xPC Target network boot image consists of running `xpcnetboot` to start the network boot server process.
- Creating xPC Target DOS Loader files consists of creating the files in a designated area. You can then copy the files to the target computer flash disk.

If you update the environment, you need to update the target boot floppy, CD boot image, network boot image, or DOS Loader files for the new xPC Target environment with the function `xpcbootdisk`.

Examples

To create a boot floppy disk, in the MATLAB window, type:

```
xpcbootdisk
```

See Also

`setxpcenv | getxpcenv`

How To

- "Target Boot Methods"
- "Command Line Target Boot Methods"
- "Command Line Target Boot Methods: Multiple Target Computers"

xpcbytes2file

Purpose

Generate file suitable for use by From File block

Syntax

```
xpcbytes2file(filename, var1, . . .,varn)
```

Arguments

`filename`	Name of the data file from which the From File block distributes data.
`var1,. . .varn`	Column of data to be output to the model.

Description

The `xpcbytes2file` function outputs one column of `var1,. . .,varn` at every time step. All variables must have the same number of columns; the number of rows and data types can differ.

Note You might have the data organized such that a row refers to a single time step and not a column. In this case, pass to `xpcbytes2file` the transpose of the variable. To optimize file writes, organize the data in columns.

Examples

In the following example, to use the From File block to output a variable `errorval` (single precision, scalar) and `velocity` (double, width 3) at every time step, you can generate the file with the command:

```
xpcbytes2file('myfile', errorval, velocity)
```

where `errorval` has class `'single'` and dimensions [1 x N] and `velocity` has class `'double'` and dimensions [3 x N].

Set up the From File block to output

```
28 bytes
(1 * sizeof('single') + 3 * sizeof('double'))
```

at every sample time.

Purpose

Open xPC Target Explorer

Syntax

MATLAB command line

```
xpcexplr
```

Description

This tool runs on the host computer and allows you to:

- Enter and change xPC Target environment properties
- Create an xPC Target bootable image
- Download, unload, and run target applications
- Monitor signals
- Tune parameters
- Add, remove, and configure xPC Target scopes
- Browse the target file system

See Also

`setxpcenv` | `getxpcenv` | `xpcbootdisk`

How To

- "Network Communication Setup"
- "Serial Communication Setup"
- "Target Boot Methods"

Purpose

Compiler settings for xPC Target environment

Syntax

```
type = xpcgetCC
type = xpcgetCC('Type')
[type, location] = xpcgetCC
location= xpcgetCC('Location')
xpcgetCC('supported')
xpcgetCC('installed')
[compilers] = xpcgetCC('installed')
```

Description

`type = xpcgetCC` and `type = xpcgetCC('Type')` return the compiler type in *type*.

`[type, location] = xpcgetCC` returns the compiler type and its location in *type* and *location*.

`location= xpcgetCC('Location')` returns the compiler location in *location*.

`xpcgetCC('supported')` lists supported compiler versions for the xPC Target environment.

`xpcgetCC('installed')` lists the xPC Target supported compilers installed on the current host computer

`[compilers] = xpcgetCC('installed')` returns the xPC Target supported compilers installed on the current host computer in a structure.

Examples

Return the compiler type.

```
type = xpcgetCC
```

Return the compiler type and compiler location.

```
>> [type, location] = xpcgetCC
```

Return the xPC Target supported compilers installed on the current host computer in a structure and access the structure fields

```
[compilers] = xpcgetCC('installed')

compilers =

1x3 struct array with fields:
    Type
    Name
    Location

compilers.Type

ans =

VisualC
```

See Also

xpcsetCC

xpcnetboot

Purpose

Create kernel to boot target computer over dedicated network

Syntax

MATLAB command line

```
xpcnetboot
xpcnetboot targetPCname
```

Arguments

targetPCName	Target computer name as identified in xPC Target Explorer.

Description

The `xpcnetboot` function creates an xPC Target kernel that a target computer within the same network can boot.

This function also starts the following services as server processes:

- Bootstrap protocol (`bootp`) — `xpcbootpserver.exe`
- Trivial file transfer protocol (`tftp`) — `xpctftpserver.exe`

These processes respond to network boot requests from the target computer.

`xpcnetboot` creates an xPC Target kernel for the default target computer (as identified in xPC Target Explorer).

`xpcnetboot` *`targetPCname`* creates an xPC Target kernel and waits for a request from the target computer named *`targetPCname`* (as identified in xPC Target Explorer).

Examples

In the following example, `xpcnetboot` creates an xPC Target kernel and waits for a request from the target computer, `TargetPC1`.

```
xpcnetboot TargetPC1
```

Purpose

Compiler settings for xPC Target environment

Syntax

```
xpcsetCC('setup')
xpcsetCC('location')
xpcsetCC('type')
xpcsetCC(type, location)
```

Description

`xpcsetCC('setup')` queries the host computer for installed C compilers that the xPC Target environment supports. You can then select the C compiler.

`xpcsetCC('location')` sets the compiler location.

`xpcsetCC('type')` sets the compiler type. `'type'` must be `VISUALC`, representing the

Microsoft Visual Studio® C compiler.

`xpcsetCC(type, location)` sets the compiler type and location.

See Also

`xpcgetCC`

How To

- "Command Line C Compiler Configuration"

xpctarget Package

Purpose

Package for all xPC Target MATLAB classes

Description

Use `xpctarget` package objects to access all of the MATLAB command line capabilities.

Functions

Assign these object creation functions to a MATLAB variable to get access to the properties and methods of the class.

Function	Description
`xpctarget.fs`	Create file system object
`xpctarget.ftp`	Create file transfer protocol (FTP) object
`xpctarget.targets`	Create container object to manage target computer environment collection objects
`xpctarget.xpc`	Create target object representing target application

Purpose

Stores target environment properties

Description

Each `xpctarget.env Class` object contains the environment properties for a particular target computer. A collection of these objects is stored in an `xpctarget.targets Class` object. An individual object in a collection is accessed via the `xpctarget.targets.Item (env collection object)` method.

Methods

Method	Description
`xpctarget.env.get (env object)`	Return property values for an environment object
`xpctarget.env.set (env object)`	Change property values for an environment object

Properties

The environment properties define communication between the host computer and target computer and the type of target boot floppy created during the setup process. An understanding of the environment properties will help you configure the xPC Target environment.

Tip To access a subset of these properties in xPC Target Explorer:

1. Expand a target computer node in the **Targets** pane.
2. Click the Target Properties icon  in the toolbar or double-click **Properties**.

- Host-to-Target Communication on page 60
- Target Settings on page 66
- Boot Configuration on page 70

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer. If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. Select `1200`, `2400`, `48 00`, `9600`, `19200`, `38400`, `57600`, or `115200` from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
`TcpIpTargetBusType`	Property values are `'PCI'`, `'ISA'`, and `` `USB' ``. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
`TcpIpTargetDriver`	Property values are `'3C90x'`, `` `I8254x' ``, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `` `Rhine' ``, `'RTLANCE'`, `'SMC91C9X'`, `` `USBAX772' ``, `` `USBAX172' ``, and `` `Auto' ``. Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
`TcpIpTargetISAIRQ`	Property value is `'n'`, where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`. Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`. Enter a port address greater than `20000` in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer. This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` and `'off'` (the default). Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.

Environment Property	Description
`MaxModelSize`	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`. Select `1 MB`, `4 MB`, or `16 MB` from the **Model size** list in the **Target Properties** pane of xPC Target Explorer. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. **Note** • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.

Environment Property	Description
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `` `on' `` and `` `off' `` (the default). If you create a target boot kernel when `ShowHardware` is `` `on' `` and boot the target computer with it, the kernel displays on the target monitor the index, bus, slot, function, and target driver for each Ethernet card. **Note** The host computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'Manual'`. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If `TargetRAMSizeMB` is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM or you do not want to use more than 64 MB, select `Auto`. If the target computer has more than 64 MB of RAM and you want to use more than 64 MB, select `Manual` and enter the amount of RAM installed in the target computer. **Note** The xPC Target kernel can use only 2 GB of memory.

Environment Property	Description
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. **Tip** To use all the features of the target scope, you also need to install a keyboard on the target computer.
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`CDBootImageLocation`	Location of `cdboot.iso` file for creation of CD target boot disk.
`DOSLoaderLocation`	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
`EmbeddedOption`	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.
`TargetBoot`	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer. If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of `Stand Alone`. **Tip** Click the **Create boot disk** button to create a bootable image in the specified boot mode.
`TargetMACAddress`	Physical target computer MAC address from which to accept boot

Environment Property	Description
	requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Purpose

Return target environment property values

Syntax

MATLAB command line

```
get(env_object)
get(env_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
env_object.get('property_name1', 'property_value1')
get(env_object, property_name_vector, property_value_vector)
env_object.property_name = property_value
```

Arguments

`env_object`	Name of a target environment object.
`'property_name'`	Name of a target environment object property. Always use quotation marks.
`property_value`	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.
`parameter_name`	The letter p followed by the parameter index. For example, `p0`, `p1`, `p2`.

Description

Not all properties are user writable. Get an individual environment object via the `xpctarget.targets.Item (env collection object)` method.

The environment properties for a target environment object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Tip To access a subset of these properties in xPC Target Explorer:

1. Expand a target computer node in the **Targets** pane.
2. Click the Target Properties icon  in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 2-74
- "Target Settings" on page 2-80
- "Boot Configuration" on page 2-84

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer. If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. Select `1200`, `2400`, `48 00`, `9600`, `19200`, `38400`, `57600`, or `115200` from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
`TcpIpTargetBusType`	Property values are `'PCI'`, `'ISA'`, and `` `USB' ``. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
`TcpIpTargetDriver`	Property values are `'3C90x'`, `` `I8254x' ``, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `` `Rhine' ``, `'RTLANCE'`, `'SMC91C9X'`, `` `USBAX772' ``, `` `USBAX172' ``, and `` `Auto' ``. Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
`TcpIpTargetISAIRQ`	Property value is `'n'`, where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`. Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`. Enter a port address greater than `20000` in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer. This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` and `'off'` (the default). Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.

Environment Property	Description
`MaxModelSize`	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`. Select `1 MB`, `4 MB`, or `16 MB` from the **Model size** list in the **Target Properties** pane of xPC Target Explorer. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. **Note** • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.

Environment Property	Description
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `` `on' `` and `` `off' `` (the default). If you create a target boot kernel when `ShowHardware` is `` `on' `` and boot the target computer with it, the kernel displays on the target monitor the index, bus, slot, function, and target driver for each Ethernet card. **Note** The host computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'Manual'`. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If `TargetRAMSizeMB` is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM or you do not want to use more than 64 MB, select `Auto`. If the target computer has more than 64 MB of RAM and you want to use more than 64 MB, select `Manual` and enter the amount of RAM installed in the target computer. **Note** The xPC Target kernel can use only 2 GB of memory.

Environment Property	Description
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. **Tip** To use all the features of the target scope, you also need to install a keyboard on the target computer.
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`CDBootImageLocation`	Location of `cdboot.iso` file for creation of CD target boot disk.
`DOSLoaderLocation`	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
`EmbeddedOption`	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.
`TargetBoot`	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer. If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of `Stand Alone`. **Tip** Click the **Create boot disk** button to create a bootable image in the specified boot mode.
`TargetMACAddress`	Physical target computer MAC address from which to accept boot

Environment Property	Description
	requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

See Also

`xpctarget.env.set (env object)`

Purpose

Change target environment object property values

Syntax

MATLAB command line

```
set(env_object)
set(env_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
env_object.set('property_name1', 'property_value1')
set(env_object, property_name_vector,
property_value_vector)
env_object.property_name = property_value
```

Arguments

`env_object`	Name of a target environment object.
`'property_name'`	Name of a target environment object property. Always use quotation marks.
`property_value`	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

Not all properties are user writable. Get an individual environment object via the `xpctarget.targets.Item (env collection object)` method.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`. The writable properties for a target environment object are listed in the following table. This table includes a description of the properties.

Tip To access a subset of these properties in xPC Target Explorer:

1. Expand a target computer node in the **Targets** pane.
2. Click the Target Properties icon  in the toolbar or double-click **Properties**.

- "Host-to-Target Communication" on page 2-88
- "Target Settings" on page 2-94
- "Boot Configuration" on page 2-98

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of xPC Target Explorer. If you select `RS-232`, you also must set the property `RS232HostPort`. If you select `TCP/IP`, then you also need to set all properties that start with `TcpIp`.

Environment Property	Description
	Note RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.
`RS232Baudrate`	Property values are `'115200'`, `'57600'`, `'38400'`, `'19200'`, `'9600'`, `'4800'`, `'2400'`, and `'1200'`. Select `1200`, `2400`, `48 00`, `9600`, `19200`, `38400`, `57600`, or `115200` from the **Baud rate** list in the **Target Properties** pane of xPC Target Explorer.
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** pane of xPC Target Explorer. The software automatically determines the COM port on the target computer. Before you can select an RS-232 port, you need to set the `HostTargetComm` property to `RS232`.

Environment Property	Description
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `255.255.255.255`, which means that a gateway is not used to connect to the target computer. If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.

Environment Property	Description
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of xPC Target Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.
`TcpIpTargetBusType`	Property values are `'PCI'`, `'ISA'`, and `` `USB' ``. Select `PCI`, `ISA`, or `USB` from the **Bus type** list in the **Target Properties** pane of xPC Target Explorer. This property is set by default to `PCI`, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer. If `TcpIpTargetBusType` is set to `PCI`, then the properties `TcpIpISAMemPort` and `TcpIpISAIRQ` have no effect on TCP/IP communication. If you are using an ISA bus card, set `TcpIpTargetBusType` to `ISA` and enter values for `TcpIpISAMemPort` and `TcpIpISAIRQ`.

Environment Property	Description
`TcpIpTargetDriver`	Property values are `'3C90x'`, `` `I8254x' ``, `'I82559'`, `'NE2000'`, `'NS83815'`, `'R8139'`, `'R8168'`, `` `Rhine' ``, `'RTLANCE'`, `'SMC91C9X'`, `` `USBAX772' ``, `` `USBAX172' ``, and `` `Auto' ``. Select `THREECOM_3C90x`, `INTEL_I8254x`, `INTEL_I82559`, `NE2000`, `NS83815`, `R8139`, `R8168`, `Rhine`, `RTLANCE`, `SMC91C9X`, `USBAX772`, `USBAX172`, or `Auto` from the **Target driver** list in the **Target Properties** pane of xPC Target Explorer.
`TcpIpTargetISAIRQ`	Property value is `'n'`, where *n* is between 5 and 15 inclusive. Select an IRQ value from the **IRQ** list in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card. On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card. Set the IRQ to 5, 10, or 11. If one of these hardware settings

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
`TcpIpTargetISAMemPort`	Property value is `'0xnnnn'`. Enter an I/O port base address in the **Address** box in the **Target Properties** pane of xPC Target Explorer. If you are using an ISA bus Ethernet card, you must enter values for the properties `TcpIpISAMemPort` and `TcpIpISAIRQ`. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card. On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card. Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.
`TcpIpTargetPort`	Property value is `'xxxxx'`. Enter a port address greater than `20000` in the **Port** box in the

Environment Property	Description
	Target Properties pane of xPC Target Explorer. This property is set by default to `22222`. The default value is higher than the reserved area (`telnet`, `ftp`, . . .) and is only of use on the target computer.

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is 0. If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` and `'off'` (the default). Set this value to `'on'` only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'off'`.

Environment Property	Description
`MaxModelSize`	Property values are `'1MB'` (the default), `'4MB'`, and `'16MB'`. Select `1 MB`, `4 MB`, or `16 MB` from the **Model size** list in the **Target Properties** pane of xPC Target Explorer. Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. The remaining memory is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. **Note** • BootFloppy and DOSLoader modes ignore this value. • In StandAlone mode, you can only use `MaxModelSize` values `'1MB'` and `'4MB'`.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of xPC Target Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.

Environment Property	Description
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box will slow the performance of your target computer.
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `` `on' `` and `` `off' `` (the default). If you create a target boot kernel when `ShowHardware` is `` `on' `` and boot the target computer with it, the kernel displays on the target monitor the index, bus, slot, function, and target driver for each Ethernet card. **Note** The host computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'Manual'`. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of xPC Target Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap. If `TargetRAMSizeMB` is set to `Auto`, the target application automatically determines the amount of memory up to 64 MB. If the target computer does not contain more than 64 MB of RAM or you do not want to use more than 64 MB, select `Auto`. If the target computer has more than 64 MB of RAM and you want to use more than 64 MB, select `Manual` and enter the amount of RAM installed in the target computer. **Note** The xPC Target kernel can use only 2 GB of memory.

Environment Property	Description
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of xPC Target Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. **Tip** To use all the features of the target scope, you also need to install a keyboard on the target computer.
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of xPC Target Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

Environment Property	Description
`BootFloppyLocation`	Drive name for creation of target boot disk.
`CDBootImageLocation`	Location of `cdboot.iso` file for creation of CD target boot disk.
`DOSLoaderLocation`	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
`EmbeddedOption`	Property values are `'Disabled'` and `'Enabled'`. This property is read only. Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.
`TargetBoot`	Property values are `'BootFloppy'`, `'CDBoot'`, `'DOSLoader'`, `'NetworkBoot'`, and `'StandAlone'`. Select `Removable Disk`, `CD`, `DOS Loader`, `Network`, or `Stand Alone` from the **Boot mode** list in the **Target Properties** pane of xPC Target Explorer. If your license file does not include the license for the xPC Target Embedded Option product, your only options are `Removable Disk`, `CD`, `DOS Loader`, and `Network`. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of `Stand Alone`. **Tip** Click the **Create boot disk** button to create a bootable image in the specified boot mode.
`TargetMACAddress`	Physical target computer MAC address from which to accept boot

Environment Property	Description
	requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons: `xx:xx:xx:xx:xx:xx` To update the MAC address in xPC Target Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

See Also

`xpctarget.env.get (env object)`

Purpose

Manage the directories and files on the target computer

Description

This class includes the directory methods from `xpctarget.fsbase Class` and implements file access methods used on the target computer.

Constructor

Constructor	Description
`xpctarget.fs`	Create file system object

Methods

These methods are inherited from `xpctarget.fsbase Class`.

Method	Description
`xpctarget.fsbase.cd`	Change folder on target computer
`xpctarget.fsbase.dir`	List contents of current folder on target computer
`xpctarget.fsbase.mkdir`	Make folder on target computer
`xpctarget.fsbase.pwd`	Current folder path of target computer
`xpctarget.fsbase.rmdir`	Remove folder from target computer

These methods are specific to class `fs`.

Method	Description
`xpctarget.fs.diskinfo`	Information about target computer drive
`xpctarget.fs.fclose`	Close open target computer file(s)
`xpctarget.fs.fileinfo`	Target computer file information
`xpctarget.fs.filetable`	Information about open files in target computer file system
`xpctarget.fs.fopen`	Open target computer file for reading

Method	Description
xpctarget.fs.fread	Read open target computer file
xpctarget.fs.fwrite	Write binary data to open target computer file
xpctarget.fs.getfilesize	Size of file on target computer
xpctarget.fs.removefile	Remove file from target computer

Purpose

Create xPC Target file system object

Syntax

MATLAB command line

```
filesys_object = xpctarget.fs('mode', 'arg1', 'arg2')
```

Arguments

`filesys_object` Variable name to reference the file system object.

`mode` Optionally, enter the communication mode:

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

`TCPIP`	Specify TCP/IP connection with target computer.
`RS232`	Specify RS-232 connection with target computer.

`arg1` Optionally, enter an argument based on the `mode` value:

IP address	If `mode` is `'TCPIP'`, enter the IP address of the target computer.
COM port	If `mode` is `'RS232'`, enter the host COM port.

`arg2` Optionally, enter an argument based on the `mode` value:

Port	If `mode` is `'TCPIP'`, enter the port number for the target computer.
Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target computer.

Description

Constructor of a file system object (`xpctarget.fs Class`). The file system object represents the file system on the target computer. You work with the file system by changing the file system object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
filesys_object=xpctarget.fs
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file system object, use the following syntax to create the additional file system objects.

```
filesys_object=xpctarget.fs('mode', 'arg1', 'arg2')
```

Examples

In the following example, a file system object for a target computer with an RS-232 connection is created.

```
fs1=xpctarget.fs('RS232','COM1','115200')

fs1 =
 xpctarget.fs
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.fs` object by passing the `xpctarget.xpc` object variable to the `xpctarget.fs` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232','COM1','115200');
>> fs2=xpctarget.fs(tg1)

fs2 =

 xpctarget.fs
```

Purpose

Information about target computer drive

Syntax

MATLAB command line

```
diskinfo(filesys_obj,target_PC_drive)
filesys_obj.diskinfo(target_PC_drive)
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.
`target_PC_drive`	Name of the target computer drive for which to return information.

Description

Method of `xpctarget.fs` objects. From the host computer, returns disk information for the specified target computer drive.

Examples

Return disk information for the target computer C:\ drive for the file system object `fsys`.

```
diskinfo(fsys,'C:\') or fsys.diskinfo('C:\')
ans =
                  Label: 'SYSTEM '
             DriveLetter: 'C'
                Reserved: ''
            SerialNumber: 1.0294e+009
    FirstPhysicalSector: 63
                FATType: 32
                FATCount: 2
           MaxDirEntries: 0
          BytesPerSector: 512
        SectorsPerCluster: 4
            TotalClusters: 2040293
              BadClusters: 0
             FreeClusters: 1007937
                    Files: 19968
               FileChains: 22480
               FreeChains: 1300
         LargestFreeChain: 64349
```

Purpose

Close open target computer file(s)

Syntax

MATLAB command line

```
fclose(filesys_obj,file_ID)
filesys_obj.fclose(file_ID)
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.
`file_ID`	File identifier of the file to close.

Description

Method of `xpctarget.fs` objects. From the host computer, closes one or more open files in the target computer file system (except standard input, output, and error). The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen` and `xpctarget.fs.filetable`). You cannot have more than eight files open in the file system.

Examples

Close the open file identified by the file identifier `h` in the file system object `fsys`.

```
fclose(fsys,h) or fsys.fclose(h)
```

See Also

`fclose` | `xpctarget.fs.fopen` | `xpctarget.fs.fread` | `xpctarget.fs.filetable` | `xpctarget.fs.fwrite`

xpctarget.fs.fileinfo

Purpose

Target computer file information

Syntax

MATLAB command line

```
fileinfo(filesys_obj,file_ID)
filesys_obj.fileinfo(file_ID)
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.
`file_ID`	File identifier of the file for which to get file information.

Description

Method of `xpctarget.fs` objects. From the host computer, gets the information for the file associated with `file_ID`.

Examples

Return file information for the file associated with the file identifier `h` in the file system object `fsys`.

```
fileinfo(fsys,h) or fsys.fileinfo(h)
ans =
               FilePos: 0
         AllocatedSize: 12288
         ClusterChains: 1
    VolumeSerialNumber: 1.0450e+009
              FullName: 'C:\DATA.DAT'
```

Purpose

Information about open files in target computer file system

Syntax

MATLAB command line

```
filetable(filesys_obj)
filesys_obj.filetable
```

Arguments

`filesys_obj`	Name of the `xpctarget.fs` file system object.

Description

Method of `xpctarget.fs` objects. From the host computer, displays a table of the open files in the target computer file system. You cannot have more than eight files open in the file system.

Examples

Return a table of the open files in the target computer file system for the file system object `fsys`.

```
filetable(fsys) or fsys.filetable
ans =
Index    Handle  Flags     FilePos  Name
-----------------------------------------
    0  00060000  R__          8512  C:\DATA.DAT
    1  00080001  R__             0  C:\DATA1.DAT
    2  000A0002  R__          8512  C:\DATA2.DAT
    3  000C0003  R__          8512  C:\DATA3.DAT
    4  001E000S  R__             0  C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a handle to one that other `xpctarget.fs` methods, such as `fclose`, can use, use the `hex2dec` function.

```
h1 = hex2dec('001E0001'))
h1 =
1966081
```

To close that file, use the `xpctarget.fs` `fclose` method.

```
fsys.fclose(h1);
```

See Also

xpctarget.fs.fopen | xpctarget.fs.fclose

Purpose

Open target computer file for reading

Syntax

MATLAB command line

```
file_ID = fopen(file_obj,'file_name')
file_ID = file_obj.fopen('file_name')
file_ID = fopen(file_obj,'file_name',permission)
file_ID = file_obj.fopen('file_name',permission)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`'file_name'`	Name of the target computer to open.
`permission`	Values are `'r'`, `'w'`, `'a'`, `'r+'`, `'w+'`, or `'a+'`. This argument is optional with `'r'` as the default value.

Description

Method of `xpctarget.fs` objects. From the host computer, opens the specified filename on the target computer for binary access.

The permission argument values are

- `'r'`

  Open the file for reading (default). The method does nothing if the file does not already exist.

- `'w'`

  Open the file for writing. The method creates the file if it does not already exist.

- `'a'`

  Open the file for appending to the file. Initially, the file pointer is at the end of the file. The method creates the file if it does not already exist.

- `'r+'`

Open the file for reading and writing. Initially, the file pointer is at the beginning of the file. The method does nothing if the file does not already exist.

- `'w+'`

  Open the file for reading and writing. The method empties the file first, if the file already exists and has content, and places the file pointer at the beginning of the file. The method creates the file if it does not already exist.

- `'a+'`

  Open the file for reading and appending to the file. Initially, the file pointer is at the beginning of the file. The method creates the file if it does not already exist.

You cannot have more than eight files open in the file system. This method returns the file identifier for the open file in `file_ID`. You use `file_ID` as the first argument to the other file I/O methods (such as `xpctarget.fs.fclose`, `xpctarget.fs.fread`, and `xpctarget.fs.fwrite`).

Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
     2883584
d = fread(h);
```

See Also

`fopen` | `xpctarget.fs.fclose` | `xpctarget.fs.fread` | `xpctarget.fs.fwrite`

Purpose

Read open target computer file

Syntax

MATLAB command line

```
A = fread(file_obj,file_ID)
A = file_obj.fread(file_ID)
A = fread(file_obj, file_ID, offset, numbytes)
A = file_obj.fread(file_ID, offset, numbytes)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`file_ID`	File identifier of the file to read.
`offset`	Position from the beginning of the file from which `fread` can start to read.
`numbytes`	Maximum number of bytes `fread` can read.

Description

Method of `xpctarget.fs` objects. From the host computer, `A = fread(file_obj,file_ID)` or `A = file_obj.fread(file_ID)` reads all the binary data from the file on the target computer and writes it into matrix `A`. The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen`).

From the host computer, `A = fread(file_obj, file_ID, offset, numbytes)` or `A = file_obj.fread(file_ID, offset, numbytes)` reads a block of bytes from `file_ID` and writes the block into matrix `A`. The `offset` argument specifies the position from the beginning of the file from which this function can start to read. `numbytes` specifies the maximum number of bytes to read. To get a count of the total number of bytes read into `A`, use the following:

```
count = length(A);
```

`length(A)` might be less than the number of bytes requested if that number of bytes are not currently available. It is zero if the operation reaches the end of the file.

Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
     2883584
d = fread(h);
```

This reads the file `data.dat` and stores all of the contents of the file to `d`. This content is in the xPC Target file format.

See Also

`fread | xpctarget.fs.fclose | xpctarget.fs.fopen | xpctarget.fs.fwrite`

Purpose

Write binary data to open target computer file

Syntax

MATLAB command line

```
fwrite(file_obj,file_ID,A)
file_obj.fwrite(file_ID,A)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`file_ID`	File identifier of the file to write.
`A`	Elements of matrix `A` to be written to the specified file.

Description

Method of `xpctarget.fs` objects. From the host computer, writes the elements of matrix `A` to the file identified by `file_ID`. The data is written to the file in column order. The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen`). `fwrite` requires that the file be open with write permission.

Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for writing.

```
h = fopen(fsys,'data.dat','w')
```

or

```
fsys.fopen('data.dat','w')

ans =
     2883584
d = fwrite(fsys,h,magic(5));
```

This writes the elements of matrix `A` to the file handle h. This content is written in column order.

See Also

`fwrite | xpctarget.fs.fclose | xpctarget.fs.fopen | xpctarget.fs.fread`

xpctarget.fs.getfilesize

Purpose

Size of file on target computer

Syntax

MATLAB command line

```
getfilesize(file_obj,file_ID)
file_obj.getfilesize(file_ID)
```

Arguments

`file_obj`	Name of the `xpctarget.fs` object.
`file_ID`	File identifier of the file to get the size of.

Description

Method of `xpctarget.fs` objects. From the host computer, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target computer file system. Use the xPC Target file object method `xpctarget.fs.fopen` to open the file system object.

Examples

Get the size of the file identifier `h` for the file system object `fsys`.

```
getfilesize(fsys,h) or fsys.getfilesize(h)
```

See Also

`xpctarget.fs.fopen`

Purpose

Interpret raw data from xPC Target file format

Syntax

```
file=readxpcfile(data)
readxpcfile('filename')
```

Arguments

`data`	Vector of `uint8` bytes.
`'filename'`	File from which the vector of `uint8` bytes is read. Vector is written

Description

The `readxpcfile` function converts xPC Target file format content (in bytes) to double precision data. A file scope creates the data.

The `readxpcfile` function returns a structure that contains the following fields:

- `version`

  Not used.

- `sector`

  Not used.

- `headersize`

  Not used.

- `numSignals`

  Array of signal names.

- `data`

  Array of signal data.

- `signalNames`

  Cell array of signal names.

After you download the data from a target computer, use one of the following to read the data:

- To read the data after you download it to the target computer, use the `fread` function
- To download the data to the target computer and read it, use the `xpctarget.fs.fread` method.

`file=readxpcfile(data)` converts `data` to double precision data representing the signals and timestamps.

`readxpcfile('filename')` converts contents of `'filename'` to double precision data representing the signals and timestamps.

Examples

Use the `xpctarget.fs` object to convert data:

```
file=xpctarget.fs;
h=file.fopen('filename');
data=file.fread(h);
file.fclose(h);
file = readxpcfile(data);
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
datafile = readxpcfile('filename') % Convert the data
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data separately.

```
xpcftp=xpctarget.ftp
xpcftp.get('filename')
handle=fopen('filename')
data=fread(handle,'*uint8'); % Data should be read in uint8 format
fclose(handle);
data=data';
datafile = readxpcfile(data); % Convert the data
```

See Also

xpctarget.ftp.get (ftp) | xpctarget.fs.fopen | xpctarget.fs.fread

xpctarget.fs.removefile

Purpose

Remove file from target computer

Syntax

MATLAB command line

```
removefile(file_obj,file_name)
file_obj.removefile(file_name)
```

Arguments

`file_name`	Name of the file to remove from the target computer file system.
`file_obj`	Name of the `xpctarget.fs` object.

Description

Method of `xpctarget.fs` objects. Removes a file from the target computer file system.

Note You cannot recover this file once it is removed.

Examples

Remove the file `data2.dat` from the target computer file system `fsys`.

```
removefile(fsys,'data2.dat')
```

or

```
fsys.removefile('data2.dat')
```

Purpose

Select target computer drive

Syntax

MATLAB command line

```
selectdrive(file_obj,'drive')
file_obj.selectdrive('drive')
```

Arguments

`drive`	Name of the drive to set.
`file_obj`	Name of the `xpctarget.fs` object.

Description

Method of `xpctarget.fs` objects. `selectdrive` sets the current drive of the target computer to the specified string. Enter the drive string with an extra backslash (\). For example, `D:\\` for the D:\ drive.

Note Use the `xpctarget.fsbase.cd` method instead to get the same behavior.

Examples

Set the current target computer drive to D:\.

```
selectdrive(fsys,'D:\\')
```

or

```
fsys.selectdrive('D:\\')
```

xpctarget.fsbase Class

Purpose

Base class of file system and file transfer protocol (FTP) classes

Description

This class is the base class for `xpctarget.fs Class` and `xpctarget.ftp Class`. All methods are inherited by the derived classes. The constructor for this class is called implicitly when the constructors for the derived classes are called:

Methods

These methods are inherited by the derived classes.

Method	Description
`xpctarget.fsbase.cd`	Change folder on target computer
`xpctarget.fsbase.dir`	List contents of current folder on target computer
`xpctarget.fsbase.mkdir`	Make folder on target computer
`xpctarget.fsbase.pwd`	Current folder path of target computer
`xpctarget.fsbase.rmdir`	Remove folder from target computer

Purpose

Change folder on target computer

Syntax

MATLAB command line

```
cd(file_obj,target_PC_dir)
file_obj.cd(target_PC_dir)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.
`target_PC_dir`	Name of the target computer folder to change to.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, changes folder on the target computer.

Examples

Change folder from the current to one named `logs` for the file system object `fsys`.

```
cd(fsys,logs) or fsys.cd(logs)
```

Change folder from the current to one named `logs` for the FTP object `f`.

```
cd(f,logs) or f.cd(logs)
```

See Also

`cd` | `xpctarget.fsbase.mkdir` | `xpctarget.fsbase.pwd`

xpctarget.fsbase.dir

Purpose

List contents of current folder on target computer

Syntax

MATLAB command line

```
dir(file_obj)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, lists the contents of the current folder on the target computer.

To get the results in an M-by-1 structure, use a syntax like `ans=dir(file_obj)`. This syntax returns a structure like the following:

```
ans =
1x5 struct array with fields:
name
date
time
bytes
isdir
```

where

- `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.
- `date` — Date of the last save of that object
- `time` — Time of the last save of that object
- `bytes` — Size in bytes of that object
- `isdir` — Logical value indicating that the object is (`1`) or is not (`0`) a folder

Examples

List the contents of the current folder for the file system object `fsys`. You can also list the contents of the current folder for the FTP object `f`.

```
dir(fsys) or dir(f)
4/12/1998     20:00              222390        IO  SYS
 11/2/2003    13:54                   6     MSDOS  SYS
 11/5/1998    20:01               93880   COMMAND  COM
 11/2/2003    13:54  <DIR>            0      TEMP
 11/2/2003    14:00                   33 AUTOEXEC  BAT
  11/2/2003   14:00                  512 BOOTSECT  DOS
  18/2/2003   16:33                 4512 SC1SIGNA  DAT
 18/2/2003    16:17  <DIR>            0     FOUND  000
 29/3/2003    19:19                 8512      DATA  DAT
 28/3/2003    16:41                 8512 DATADATA  DAT
 28/3/2003    16:29                 4512 SC4INTEG  DAT
  1/4/2003     9:28           201326592 PAGEFILE  SYS
 11/2/2003    14:13  <DIR>            0     WINNT
    4/5/2001  13:05                214432 NTLDR       '
  4/5/2001    13:05                 34468 NTDETECT  COM
 11/2/2003    14:15  <DIR>            0   DRIVERS
  22/1/2001   11:42                  217    BOOT     INI'
 28/3/2003    16:41                 8512         A  DAT
 29/3/2003    19:19                 2512 SC3SIGNA  DAT
 11/2/2003    14:25  <DIR>            0   INETPUB
 11/2/2003    14:28                    0    CONFIG  SYS
 29/3/2003    19:10                 2512 SC3INTEG  DAT
  1/4/2003    18:05                 2512   SC1GAIN  DAT
   11/2/2003  17:26  <DIR>            0 UTILIT~1
```

You must use the `dir(f)` syntax to list the contents of the folder.

See Also

`dir` | `xpctarget.fsbase.mkdir` | `xpctarget.fsbase.cd` | `xpctarget.fsbase.pwd`

xpctarget.fsbase.mkdir

Purpose

Make folder on target computer

Syntax

MATLAB command line

```
mkdir(file_obj,dir_name)
file_obj.mkdir(dir_name)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.
`dir_name`	Name of the folder to be created.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, makes a new folder in the current folder on the target computer file system.

Note that to delete a folder from the target computer, you need to reboot the PC into DOS or some other operating system and use a utility in that system to delete the folder.

Examples

Create a new folder, `logs`, in the target computer file system object `fsys`.

```
mkdir(fsys,logs)
```

or

```
fsys.mkdir(logs)
```

Create a new folder, `logs`, in the target computer FTP object `f`.

mkdir(f,logs) or f.mkdir(logs)

See Also

`mkdir` | `xpctarget.fsbase.dir` | `xpctarget.fsbase.pwd`

Purpose

Current folder path of target computer

Syntax

MATLAB command line

```
pwd(file_obj)
file_obj.pwd
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` or `xpctarget.fs` object.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. Returns the pathname of the current target computer folder.

Examples

Return the target computer current folder for the file system object `fsys`.

```
pwd(fsys) or fsys.pwd
```

Return the target computer current folder for the FTP object `f`.

pwd(f) or f.pwd

See Also

`pwd` | `xpctarget.fsbase.dir` | `xpctarget.fsbase.mkdir`

xpctarget.fsbase.rmdir

Purpose

Remove folder from target computer

Syntax

MATLAB command line

```
rmdir(file_obj,dir_name)
file_obj.rmdir(dir_name)
```

Arguments

`dir_name`	Name of the folder to remove from the target computer file system.
`file_obj`	Name of the `xpctarget.fs` object.

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. Removes a folder from the target computer file system.

Note You cannot recover this folder once it is removed.

Examples

Remove the folder `data2dir.dat` from the target computer file system `fsys`.

```
rmdir(f,'data2dir.dat')
```

or

```
fsys.rmdir('data2dir.dat')
```

Purpose

Manage the directories and files on the target computer via file transfer protocol (FTP)

Description

The FTP object represents the file on the target computer. You work with the file directories using the inherited methods, and transport the file between the host and target computers using the `xpctarget.ftp` methods.

Constructor

Constructor	Description
`xpctarget.ftp`	Create file transfer protocol (FTP) object

Methods

These methods are inherited from `xpctarget.fsbase Class`.

Method	Description
`xpctarget.fsbase.cd`	Change folder on target computer
`xpctarget.fsbase.dir`	List contents of current folder on target computer
`xpctarget.fsbase.mkdir`	Make folder on target computer
`xpctarget.fsbase.pwd`	Current folder path of target computer
`xpctarget.fsbase.rmdir`	Remove folder from target computer

These methods are specific to class `ftp`.

Method	Description
`xpctarget.ftp.get (ftp)`	Retrieve copy of requested file from target computer
`xpctarget.ftp.put`	Copy file from host computer to target computer

Purpose

Create file transfer protocol (FTP) object

Syntax

MATLAB command line

```
file_object = xpctarget.ftp('mode', 'arg1', 'arg2')
```

Arguments

`file_object` Variable name to reference the FTP object.

`mode` Optionally, enter the communication mode:

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

`TCPIP`	Specify TCP/IP connection with target computer.
`RS232`	Specify RS-232 connection with target computer.

`arg1` Optionally, enter an argument based on the `mode` value:

IP address	If `mode` is `'TCPIP'`, enter the IP address of the target computer.
COM port	If `mode` is `'RS232'`, enter the host COM port.

`arg2` Optionally, enter an argument based on the `mode` value:

Port	If `mode` is `'TCPIP'`, enter the port number for the target computer.
Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target computer.

Description

Constructor of an FTP object (`xpctarget.ftp Class`). The FTP object represents the file on the target computer. You work with the file by changing the file object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
file_object=xpctarget.ftp
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file object, use the following syntax to create the additional file objects.

```
file_object=xpctarget.ftp('mode', 'arg1', 'arg2')
```

Examples

In the following example, a file object for a target computer with an RS-232 connection is created.

```
f=xpctarget.ftp('RS232','COM1','115200')

f =
 xpctarget.ftp
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.ftp` object by passing the `xpctarget.xpc` object variable to the `xpctarget.ftp` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232','COM1','115200');
>> f2=xpctarget.ftp(tg1)

f2 =

 xpctarget.ftp
```

xpctarget.ftp.get (ftp)

Purpose

Retrieve copy of requested file from target computer

Syntax

MATLAB command line

```
get(file_obj,file_name)
file_obj.get(file_name)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` object.
`file_name`	Name of a file on the target computer.

Description

Method of `xpctarget.ftp` objects. Copies the specified filename from the target computer to the current folder of the host computer. `file_name` must be either a fully qualified filename on the target computer, or located in the current folder of the target computer.

Examples

Retrieve a copy of the file named `data.dat` from the current folder of the target computer file object `f`.

```
get(f,'data.dat') or f.get('data.dat')
ans = data.dat
```

See Also

`xpctarget.ftp.put`

Purpose

Copy file from host computer to target computer

Syntax

MATLAB command line

```
put(file_obj,file_name)
file_obj.put(file_name)
```

Arguments

`file_obj`	Name of the `xpctarget.ftp` object.
`file_name`	Name of the file to copy to the target computer.

Description

Method of `xpctarget.ftp` objects. Copies a file from the host computer to the target computer. `file_name` must be a file in the current folder of the host computer. The method writes `file_name` to the target computer disk.

`put` might be slower than the `get` operation for the same file. This is expected behavior.

Examples

Copy the file `data2.dat` from the current folder of the host computer to the current folder of the target computer FTP object `f`.

```
put(f,'data2.dat')
```

or

```
fsys.put('data2.dat')
```

See Also

`xpctarget.fsbase.dir` | `xpctarget.ftp.get (ftp)`

xpctarget.targets Class

Purpose

Container object to manage target computer environment collection objects

Description

The targets class contains a collection of environment settings, stored in `xpctarget.env Class` objects.

Constructor

Constructor	Description
`xpctarget.targets`	Create container object to manage target computer environment collection objects

Methods

Method	Description
`xpctarget.targets.Add (env collection object)`	Add a new xPC Target environment collection object.
`xpctarget.targets.getTargetNames (env collection object)`	Retrieve all xPC Target environment collection object names.
`xpctarget.targets.Item (env collection object)`	Retrieve xPC Target environment collection object.
`xpctarget.targets.makeDefault (env collection object)`	Set target computer environment collection object as default.
`xpctarget.targets.Remove (env collection object)`	Remove an xPC Target environment collection object.

Properties

Property	Description	Writable
`DefaultTarget`	Returns an `xpctarget.env` object that references the default target computer object environment.	No
`NumTargets`	Returns the number of target computer environment objects in the container.	No

xpctarget.targets

Purpose

Create container object to manage target computer environment collection objects

Syntax

MATLAB command line

```
env_collection_object = xpctarget.targets
```

Description

Constructor for target environment object collection (`xpctarget.targets Class`). The collection manages the environment object (`xpctarget.env Class`) for a multitarget xPC Target system. (This is in contrast to the `setxpcenv` and `getxpcenv` functions, which manage the environment properties for the default target computer.) You work with the environment objects by changing the environment properties using methods.

Use the syntax

```
env_object = xpctarget.targets
```

Access properties of an `env_collection_object` object with `env_collection_object.propertyname`, `env_collection_object.propertyname.propertyname`, or with the `xpctarget.targets.get (env collection object)` and `xpctarget.targets.set (env collection object)` commands.

Access an individual environment object via `xpctarget.targets.Item (env collection object)`,

Examples

Create an environment container object. With this object, you can manage the environment collection objects for all the targets in your system.

```
tgs=xpctarget.targets
```

See Also

`xpctarget.targets.get (env collection object)` | `xpctarget.targets.set (env collection object)`

xpctarget.targets.Add (env collection object)

Purpose

Add new xPC Target environment collection object

Syntax

MATLAB command line

```
env_collection_object.Add
```

Description

Method of `xpctarget.targets` objects. `Add` creates an xPC Target environment collection object on the host computer.

Examples

Add a new xPC Target environment collection object to the system. Assume that `tgs` represents the environment collection object. The first `get(tgs)` function returns the current number of target computers. The second function returns the number of target computers after you add one.

```
tgs=xpctarget.targets;
get(tgs);
tgs.Add;
get(tgs);
```

See Also

`xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

xpctarget.targets.get (env collection object)

Purpose

Return target object collection environment property values

Syntax

MATLAB command line

```
get(env_collection_object, 'env_collection_object_property')
```

Arguments

`env_collection_object`	Name of a collection of target objects.
`'env_collection_object_property'`	Name of a target object environment property.

Description

`get` gets the values of environment properties for a collection of target objects.

The environment properties for a target environment object collection are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
`DefaultTarget`	Contains an instance of the default target environment object (`xpctarget.env`).	No
`NumTargets`	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

Examples

List the values of all the target object collection environment property values. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
```

List the value for the target object environment collection property `NumTargets`. Note that the property name is a string, in quotation marks, and not case sensitive.

```
get(tgs,'NumTargets') or tgs.get('NumTargets'))
```

See Also

`get` | `xpctarget.targets.set (env collection object)` | `set`

xpctarget.targets.getTargetNames (env collection object)

Purpose

Retrieve xPC Target environment object names

Syntax

MATLAB command line

```
env_collection_object.getTargetNames
```

Description

Method of `xpctarget.targets` objects. `getTargetNames` retrieves the names of all existing xPC Target environment collection objects from the `xpctarget.targets` class.

Examples

Retrieve the names of all xPC Target environment collection objects in the system. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
```

See Also

`xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

Purpose

Retrieve specific xPC Target environment (env) object

Syntax

MATLAB command line

```
env_collection_object.Item('env_object_name')
```

Description

Method of `xpctarget.targets` objects. `Item` retrieves a specific environment object (`xpctarget.env Class`) from the `xpctarget.targets` class. Use this method to work with a particular target computer environment object.

Examples

Retrieve a new xPC Target environment collection object from the system. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.Item('TargetPC1')
```

See Also

`xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

xpctarget.targets.makeDefault (env collection object)

Purpose

Set specific target computer environment object as default

Syntax

MATLAB command line

```
env_collection_object.makeDefault(`env_object_name')
```

Description

Method of `xpctarget.targets` objects. `makeDefault` sets the specified target computer environment object as the default target computer from the `xpctarget.targets` class.

Examples

Set the specified target collection object as the default target computer collection. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.makeDefault('TargetPC2')
```

See Also

`xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

Purpose

Remove specific xPC Target environment object

Syntax

MATLAB command line

```
env_collection_object.Remove('env_collection_object_name')
```

Description

Method of `xpctarget.targets` objects. `Remove` removes an existing xPC Target environment object from the environment collection. If you remove the target environment object of the default target computer, the next target environment object becomes the default target computer. You can remove all but the last target computer, which becomes the default target computer.

Examples

Remove an xPC Target environment collection object from the system. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;
get(tgs);
tgs.getTargetNames
tgs.Remove('TargetPC2')
```

See Also

`xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

xpctarget.targets.set (env collection object)

Purpose

Change target object environment collection object property values

Syntax

MATLAB command line

```
set(env_collection_object)
set(env_collection_object, 'property_name1',
'property_value1','property_name2', 'property_value2', . . .)
env_collection_object.set('property_name1',
'property_value1')
set(env_collection_object, property_name_vector,
property_value_vector)
env_collection_object.property_name = property_value
```

Arguments

`env_collection_object`	Name of a target environment collection object.
`'property_name'`	Name of a target object environment collection property. Always use quotation marks.
`property_value`	Value for a target object environment collection property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

`set` sets the values of environment properties for a collection of target object environments. Not all properties are user writable.

Properties must be entered in pairs or, using the alternative syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The environment properties for a target object collection are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
`DefaultTarget`	Contains an instance of the default target environment object (`xpctarget.env`).	No
`NumTargets`	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

See Also

`get` | `set` | `xpctarget.targets.get (env collection object)`

xpctarget.xpc Class

Purpose

Target object representing target application

Description

Provides access to methods and properties used to start and stop the target application, read and set parameters, monitor signals, and retrieve status information about the target computer.

Constructor

Constructor	Description
`xpctarget.xpc`	Create target object representing target application

Methods

Method	Description
`xpctarget.xpc.addscope`	Create scopes
`xpctarget.xpc.close`	Close serial port connecting host computer with target computer
`xpctarget.xpc.get (target application object)`	Return target application object property values
`xpctarget.xpc.getlog`	All or part of output logs from target object
`xpctarget.xpc.getparam`	Value of target object parameter index
`xpctarget.xpc.getparamid`	Parameter index from parameter list
`xpctarget.xpc.getparamname`	Block path and parameter name from index list
`xpctarget.xpc.getscope`	Scope object pointing to scope defined in kernel
`xpctarget.xpc.getsignal`	Value of target object signal index
`xpctarget.xpc.getsignalid`	Signal index or signal property from signal list
`xpctarget.xpc.getsignalidfromlabel`	Return vector of signal indices
`xpctarget.xpc.getsignallabel`	Return signal label
`xpctarget.xpc.getsignalname`	Signal name from index list

Method	Description
xpctarget.xpc.getxpcpci	Determine which PCI boards are installed in target computer
xpctarget.xpc.load	Download target application to target computer
xpctarget.xpc.loadparamset	Restore parameter values saved in specified file
xpctarget.xpc.reboot	Reboot target computer
xpctarget.xpc.remscope	Remove scope from target computer
xpctarget.xpc.saveparamset	Save current target application parameter values
xpctarget.xpc.set (target application object)	Change target application object property values
xpctarget.xpc.setparam	Change writable target object parameters
xpctarget.xpc.start (target application object)	Start execution of target application on target computer
xpctarget.xpc.stop (target application object)	Stop execution of target application on target computer
xpctarget.xpc.targetping	Test communication between host and target computers
xpctarget.xpc.unload	Remove current target application from target computer

Properties

Properties are read using `xpctarget.xpc.get (target application object)`. Writable properties are written using `xpctarget.xpc.set (target application object)`.

Property	Description	Writable
`Application`	Name of the Simulink® model and target application built from that model.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions. • Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **xPC Target options** node of the model Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET • Interrupt latency required to schedule and run one step of the model	No

Property	Description	Writable
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`Connected`	Communication status between the host computer and the target computer. Values are `'Yes'` and `'No'`.	No
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Returning this status to `'none'` requires either a faster processor or a larger sample time.	No
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes

Property	Description	Writable
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No
`Mode`	Type of Simulink Coder™ code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No

Property	Description	Writable
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	No
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	No
`OutputLog`	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No

Property	Description	Writable
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "User Interaction" for limitations on target property changes to sample times.)	Yes
`Scopes`	List of index numbers, with one index for each scope.	No
`SessionTime`	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`Signals`	List of viewable signals. This list is visible only when `ShowSignals` is set to `'on'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No

Property	Description	Writable
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you need to select the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

Purpose

Create target object representing target application

Syntax

MATLAB command line

```
target_object = xpctarget.xpc('mode', 'arg1', 'arg2')
target_object=xpctarget.xpc('target_object_name')
```

Arguments

`target_object` — Variable name to reference the target object

`mode` — Optionally, enter the communication mode

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

`TCPIP`	Enable TCP/IP connection with target computer.
`RS232`	Enable RS-232 connection with target computer.

`arg1` — Optionally, enter an argument based on the `mode` value:

IP address	If `mode` is `'TCPIP'`, enter the IP address of the target computer.
COM port	If `mode` is `'RS232'`, enter the host COM port.

`arg2` — Optionally, enter an argument based on the `mode` value:

Port	If `mode` is `'TCPIP'`, enter the port number for the target computer.

	Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target computer.
`target_object_name`	Target object name as specified in the xPC Target Explorer	

Description

Constructor of a target object (`xpctarget.xpc Class`). The target object represents the target application and target computer. You make changes to the target application by changing the target object using methods and properties.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target computers in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax to construct a corresponding target object from the MATLAB Command Window.

```
target_object=xpctarget.xpc('target_object_name')
```

Examples

Before you build a target application, you can check the connection between your host and target computers by creating a target object, then using the `xpctarget.xpc.targetping` method to check the connection.

```
tg = xpctarget.xpc
 xPC Object
     Connected            = Yes
     Application          = loader

tg.targetping
```

```
ans =

success
```

If you have a second target computer for which you want to check the connection, create a second target object. In the following example, the connection with the second target computer is an RS-232 connection.

```
tg1=xpctarget.xpc('RS232','COM1','115200')

xPC Object
  Connected            = Yes
  Application          = loader
```

If you have an xPC Target Explorer target object, and you want to construct a corresponding target object in the MATLAB Command Window, use a command like the following:

```
target_object=xpctarget.xpc('TargetPC1')
```

See Also

xpctarget.xpc.get (target application object) |
xpctarget.xpc.set (target application object) |
xpctarget.xpc.targetping

Purpose

Create scopes

Syntax

MATLAB command line

Create a scope and scope object without assigning to a MATLAB variable.

```
addscope(target_object, scope_type, scope_number)
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object,
     scope_type, scope_number)
scope_object = target_object.addscope(scope_type,
     scope_number)
```

Target computer command line — When you are using this command on the target computer, you can only add a target scope.

```
addscope
addscope scope_number
```

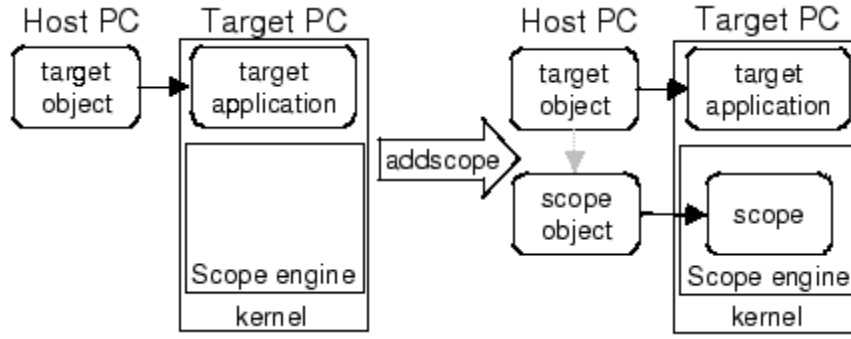
Arguments

`target_object`	Name of a target object. The default target name is `tg`.
`scope_type`	Values are `'host'`, `'target'`, or `'file'`. This argument is optional with `host` as the default value.
`scope_number`	Vector of new scope indices. This argument is optional. The next available integer in the target object property `Scopes` as the default value. If you enter a scope index for an existing scope object, the result is an error.

Description

`addscope` creates a scope of the specified type and updates the target object property `Scopes`. This method returns a scope object vector. If

the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The xPC Target product supports 10 target or host scopes, and eight file scopes, for a maximum of 28 scopes. If you try to add a scope with the same index as an existing scope, the result is an error.



Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, assigned to the variable `sc1`. The target object property `Scopes` is changed from `No scopes defined` to `1`.

```
sc1 = addscope(tg,'target',1)
```

or

```
sc1 = tg.addscope('target',1)
```

Create a scope with the method `addscope` and then create a scope object, corresponding to this scope, using the method `getscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, but it is not assigned to a variable. The target object property `Scopes` is changed from `No scopes defined` to `1`.

```
addscope(tg,'target',1) or tg.addscope('target',1)
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
```

Create two scopes using a vector of scope objects `scvector`. Two target scopes are created on the target computer with scope indices of 1 and 2, and two scope objects are created on the host computer that represent the scopes on the target computer. The target object property `Scopes` is changed from `No scopes defined` to `1,2`.

```
scvector = addscope(tg, 'target', [1, 2])
```

Create a scope and scope object `sc4 of` type `file` using the method `addscope`. A file scope is created on the target computer with an index of 4. A scope object is created on the host computer and is assigned to the variable `sc4`. The target object property `Scopes` is changed from `No scopes defined` to `4`.

```
sc4 = addscope(tg,'file',4) or sc4 = tg.addscope('file',4)
```

See Also

`xpctarget.xpc.remscope` | `xpctarget.xpc.getscope`

How To

- "Application and Driver Scripts"

xpctarget.xpc.close

Purpose

Close serial port connecting host computer with target computer

Syntax

MATLAB command line

```
close(target_object)
target_object.close
```

Arguments

`target_object`	Name of a target object.

Description

`close` closes the serial connection between the host computer and a target computer. If you want to use the serial port for another function without quitting the MATLAB window – for example, a modem – use this function to close the connection.

Purpose

Return target application object property values

Syntax

MATLAB command line

```
get(target_object, 'target_object_property')
```

Arguments

`target_object`	Name of a target object.
`'target_object_property'`	Name of a target object property.

Description

`get` gets the value of readable target object properties from a target object.

The properties for a target object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model and target application built from that model.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes.	No

Property	Description	Writable
	• Asynchronous interruptions. • Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **xPC Target options** node of the Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET • Interrupt latency required to schedule and run one step of the model	
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`Connected`	Communication status between the host computer and the target computer. Values are `'Yes'` and `'No'`.	No
`CPUoverload`	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from `'none'` to `'detected'` and the current run is stopped. Returning this status to `'none'` requires either a faster processor or a larger sample time.	No
`ExecTime`	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No

Property	Description	Writable
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes
`MaxLogSamples`	Maximum number of samples for each logged signal within the circular buffers for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog`. `StateLog` and `OutputLog` can have one or more signals. This value is calculated by dividing the **Signal Logging Buffer Size** by the number of logged signals. The **Signal Logging Buffer Size** box is in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
`MaxTET`	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
`MinTET`	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No

Property	Description	Writable
`Mode`	Type of Simulink Coder code generation. Values are `'Real-Time Singletasking'`, `'Real-Time Multitasking'`, and `'Accelerate'`. The default value is `'Real-Time Singletasking'`. Even if you select `'Real-Time Multitasking'`, the actual mode can be `'Real-Time Singletasking'`. This happens if your model contains only one or two tasks and the sample rates are equal.	No
`NumLogWraps`	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds `MaxLogSamples`.	No
`NumParameters`	The number of parameters from your Simulink model that you can tune or change.	No
`NumSignals`	The number of signals from your Simulink model that are available to be viewed with a scope.	No
`OutputLog`	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

Property	Description	Writable
`Parameters`	List of tunable parameters. This list is visible only when `ShowParameters` is set to `'on'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "User Interaction" for limitations on target property changes to sample times.)	Yes
`Scopes`	List of index numbers, with one index for each scope.	No
`SessionTime`	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes

Property	Description	Writable
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`Signals`	List of viewable signals. This list is visible only when `ShowSignals` is set to `'on'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
`Status`	Execution status of your target application. Values are `'stopped'` and `'running'`.	No
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes

Property	Description	Writable
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application. To enable logging of the TET, you need to select the **Log Task Execution Time** check box in the **xPC Target options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

Examples

List the value for the target object property `StopTime`. Notice that the property name is a string, in quotation marks, and not case sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')
ans = 0.2
```

See Also

`get` | `set` | `xpctarget.xpc.set (target application object)` | `xpctarget.xpcsc.get (scope object)` | `xpctarget.xpc.set (target application object)`

xpctarget.xpc.getlog

Purpose

All or part of output logs from target object

Syntax

MATLAB command line

```
log = getlog(target_object, 'log_name', first_point,
number_samples, decimation)
```

Arguments

`log`	User-defined MATLAB variable.
`'log_name'`	Values are `TimeLog`, `StateLog`, `OutputLog`, or `TETLog`. This argument is required.
`first_point`	First data point. The logs begin with 1. This argument is optional. Default is 1.
`number_samples`	Number of samples after the start time. This argument is optional. Default is all points in log.
`decimation`	1 returns all sample points. `n` returns every `n`th sample point. This argument is optional. Default is 1.

Description

Use this function instead of the function `get` when you want only part of the data.

Examples

To get the first 1000 points in a log,

```
Out_log = getlog(tg, 'TETLog', 1, 1000)
```

To get every other point in the output log and plot values,

```
Output_log = getlog(tg, 'TETLog', 1, 10, 2)
Time_log = getlog(tg, 'TimeLog', 1, 10, 2)
plot(Time_log, Output_log)
```

How To

- `xpctarget.xpc.get (target application object)`
- "Set Configuration Parameters"

Purpose

Value of target object parameter index

Syntax

MATLAB command line

```
getparam(target_object, parameter_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.

Description

`getparam` returns the value of the parameter associated with `parameter_index`.

Examples

Get the value of parameter index `5`.

```
getparam(tg, 5)
ans = 400
```

xpctarget.xpc.getparamid

Purpose

Parameter index from parameter list

Syntax

MATLAB command line

```
getparamid(target_object, 'block_name', 'parameter_name')
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`'block_name'`	Simulink block path without model name.
`'parameter_name'`	Name of a parameter within a Simulink block.

Description

`getparamid` returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

Examples

Get the parameter property for the parameter `Gain` in the Simulink block `Gain1`, incrementally increase the gain, and pause to observe the signal trace.

```
id = getparamid(tg, 'Subsystem/Gain1', 'Gain')
for i = 1 : 3
  set(tg, id, i*2000);
  pause(1);
end
```

Get the property index of a single block.

```
getparamid(tg, 'Gain1', 'Gain') ans = 5
```

See Also

`xpctarget.xpc.getsignalid`

How To

- “Application and Driver Scripts”

- “Why Does the getparamid Function Return Nothing?”

xpctarget.xpc.getparamname

Purpose

Block path and parameter name from index list

Syntax

MATLAB command line

```
getparamname(target_object, parameter_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.

Description

`getparamname` returns two argument strings, block path and parameter name, from the index list for the specified parameter index.

Examples

Get the block path and parameter name of parameter index `5`.

```
[blockPath,parName]=getparamname(tg,5)
blockPath =
Signal Generator
parName =
Amplitude
```

Purpose

Scope object pointing to scope defined in kernel

Syntax

MATLAB command line

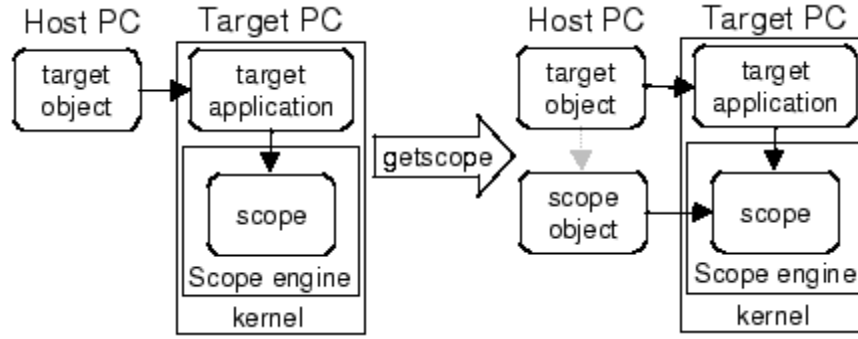
```
scope_object_vector = getscope(target_object, scope_number)
scope_object = target_object.getscope(scope_number)
```

Arguments

`target_object`	Name of a target object.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`. The vector can have only one element.
`scope_object`	MATLAB variable for a new scope object vector. The vector can have only one scope object.

Description

`getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method `get(target_object, 'scopes')` or `target_object.scopes`.



Examples

If your Simulink model has an xPC Target scope block, a target scope is created at the time the target application is downloaded to the target

computer. To change the number of samples, you need to create a scope object and then change the scope object property `NumSamples`.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
sc1.NumSample = 500
```

The following example gets the properties of all scopes on the target computer and creates a vector of scope objects on the host computer. If the target object has more than one scope, it create a vector of scope objects.

```
scvector = getscope(tg)
```

See Also

`getxpcenv` | `xpctarget.xpc.remscope`

How To

- "Application and Driver Scripts"

Purpose

Value of target object signal index

Syntax

MATLAB command line

```
getsignal(target_object, signal index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getsignal` returns the value of the signal associated with `signal_index`.

Examples

Get the value of signal index `2`.

```
getsignal(tg, 2)
ans =  -3.3869e+006
```

xpctarget.xpc.getsignalid

Purpose

Signal index or signal property from signal list

Syntax

MATLAB command line

```
getsignalid(target_object, 'signal_name')
tg.getsignalid('signal_name')
```

Arguments

`target_object`	Name of an existing target object.
`signal_name`	Enter the name of a signal from your Simulink model. For blocks with a single signal, the `signal_name` is equal to the `block_name`. For blocks with multiple signals, the xPC Target software appends S1, S2 . . . to the `block_name`.

Description

`getsignalid` returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

Examples

Get the signal index for the single signal from the Simulink block `Gain1`.

```
getsignalid(tg, 'Gain1') or tg.getsignalid('Gain1')
ans = 6
```

See Also

`xpctarget.xpc.getparamid`

How To

- "Application and Driver Scripts"
- "Why Does the getparamid Function Return Nothing?"

Purpose

Return vector of signal indices

Syntax

MATLAB command line

```
getsignalidsfromlabel(target_object, signal_label)
target_object.getsignalidsfromlabel(signal_label)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_label`	Signal label (from Simulink model).

Description

`getsignalidsfromlabel` returns a vector of one or more signal indices that are associated with the labeled signal, `signal_label`. This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the xPC Target software refers to Simulink signal names as signal labels.

Examples

Get the vector of signal indices for a signal labeled `Gain`.

```
>> tg.getsignalidsfromlabel('xpcoscGain')
ans =
0
```

See Also

`xpctarget.xpc.getsignallabel`

xpctarget.xpc.getsignallabel

Purpose

Return signal label

Syntax

MATLAB command line

```
getsignallabel(target_object, signal_index)
target_object.getsignallabel(signal_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getsignallabel` returns the signal label for the specified signal index, `signal_index`. `signal_label`. This function assumes that you have labeled the signal for which you request the label (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the xPC Target software refers to Simulink signal names as signal labels.

Examples

```
>> getsignallabel(tg, 0)
ans =
xpcoscGain
```

See Also

`xpctarget.xpc.getsignalidsfromlabel`

Purpose

Signal name from index list

Syntax

MATLAB command line

```
getsignalname(target_object, signal_index)
target_object.getsignalname(signal_index)
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`signal_index`	Index number of the signal.

Description

`getparamname` returns one argument string, signal name, from the index list for the specified signal index.

Examples

Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)
sigName =
Gain2
```

xpctarget.xpc.getxpcpci

Purpose

Determine which PCI boards are installed in target computer

Syntax

MATLAB command line

```
getxpcpci(target_object, 'type_of_boards')
getxpcpci(target_object, 'verbose')
```

Arguments

`target_object`	Variable name to reference the target object.
`type_of_boards`	Values are no arguments, `'all'`, and `'supported'`.
`verbose`	Argument to include the base address register information in the PCI device display.

Description

The `getxpcpci` function displays, in the MATLAB window, which PCI boards are installed in the target computer. By default, `getxpcpci` displays this information for the target object, `tg`. If you have multiple target computers in your system, you can call the `getxpcpci` function for a particular target object, `target_object`.

Only devices supported by driver blocks in the xPC Target block library are displayed. The information includes the PCI bus number, slot number, assigned IRQ number, manufacturer name, board name, device type, manufacturer PCI ID, base address, and the board PCI ID itself.

The following preconditions must be met before you can use this function:

- The host-target communication link must be working. (The function `xpctargetping` must return `success` before you can use the function `getxpcpci`.)
- Either a target application is loaded or the loader is active. The latter is used to query for resources assigned to a specific PCI device,

which have to be provided to a driver block dialog box before the model build process.

Examples

The following example displays the installed PCI devices, not only the devices supported by the xPC Target block library. This includes graphics controllers, network cards, SCSI cards, and even devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

```
getxpcpci('all')
```

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information.

```
getxpcpci('supported')
```

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information and base address register contents.

```
getxpcpci('verbose')
```

When called with the `'supported'` option, `getxpcpci` does not access the target computer.

To display the list of PCI devices installed on the target computer, `tg1`, first create a target object, `tg1`, for that target computer. Then, call `getxpcpci` with the `'all'` option. For example:

```
tg1=xpctarget.xpc('RS232','COM1','115200')
getxpcpci(tg1, 'all')
```

To return the result of a `getxpcpci` query in the struct `pcidevs` instead of displaying it, assign the function to `pcidevs`. The struct `pcidevs` is an array with one element for each detected PCI device. Each element combines the information by a set of field names. The struct contains more information compared to the displayed list. Its contents vary according to the options you specify for the function.

```
pcidevs = getxpcpci
```

Purpose

Download target application to target computer

Syntax

MATLAB command line

```
load(target_object,'target_application')
target_object.load('target_application')
```

Arguments

`target_object`	Name of an existing target object.
`target_application`	Simulink model and target application name.

Description

Before using this function, the target computer must be booted with the xPC Target kernel, and the target application must be built in the current working folder on the host computer.

If an application was previously loaded, the old target application is first unloaded before downloading the new target application. The method `load` is called automatically after the Simulink Coder build process.

Note If you are running in Standalone mode, this command has no effect. To load a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

Examples

Load the target application `xpcosc` represented by the target object `tg`.

```
load(tg,'xpcosc') or tg.load('xpcosc')
+tg or tg.start or start(tg)
```

See Also

`xpctarget.xpc.unload`

How To

- "Application and Driver Scripts"

xpctarget.xpc.loadparamset

Purpose

Restore parameter values saved in specified file

Syntax

MATLAB command line

```
loadparamset(target_object,'filename')
target_object.loadparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file that contains the saved parameters.

Description

`loadparamset` restores the target application parameter values saved in the file `filename`. This file must be located on a local drive of the target computer. This method assumes that you have a parameter file from a previous run of the `xpctarget.xpc.saveparamset` method.

See Also

`xpctarget.xpc.saveparamset`

Purpose

Reboot target computer

Syntax

MATLAB command line

```
reboot(target_object)
```

Target computer command line

```
reboot
```

Arguments

`target_object`	Name of an existing target object.

Description

`reboot` reboots the target computer, and if a target boot disk is still present, the xPC Target kernel is reloaded.

You can also use this method to reboot the target computer back to Windows after removing the target boot disk.

Note This method might not work on some target hardware.

See Also

`xpctarget.xpc.load` | `xpctarget.xpc.unload`

xpctarget.xpc.remscope

Purpose

Remove scope from target computer

Syntax

MATLAB command line

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)
remscope(target_object)
target_object.remscope
```

Target computer command line

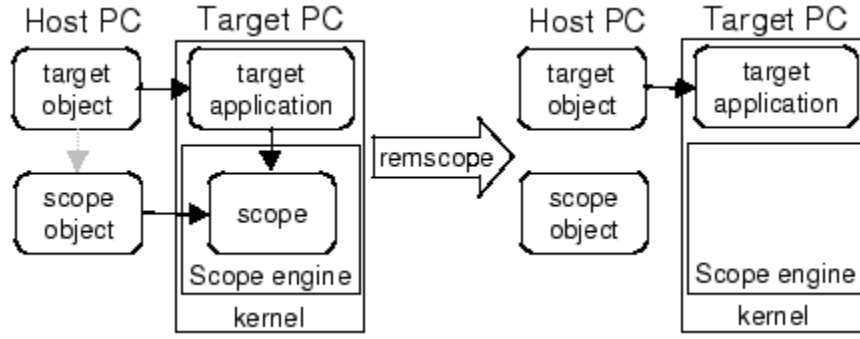
```
remscope scope_number
remscope 'all'
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.
`scope_number_vector`	Vector of existing scope indices listed in the target object property `Scopes`.
`scope_number`	Single scope index.

Description

If a scope index is not given, the method `remscope` deletes all scopes on the target computer. The method `remscope` has no return value. The scope object representing the scope on the host computer is not deleted.



Note that you can only permanently remove scopes that are added with the method `addscope`. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

Examples

Remove a single scope.

```
remscope(tg,1)
```

or

```
tg.remscope(1)
```

Remove two scopes.

```
remscope(tg,[1 2])
```

or

```
tg.remscope([1,2])
```

Remove all scopes.

```
remscope(tg)
```

or

```
tg.remscope
```

See Also

`xpctarget.xpc.addscope` | `xpctarget.xpc.getscope`

How To

- "Application and Driver Scripts"

xpctarget.xpc.saveparamset

Purpose

Save current target application parameter values

Syntax

MATLAB command line

```
saveparamset(target_object,'filename')
target_object.saveparamset('filename')
```

Arguments

`target_object`	Name of an existing target object.
`filename`	Enter the name of the file to contain the saved parameters.

Description

`saveparamset` saves the target application parameter values in the file `filename`. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the `xpctarget.xpc.loadparamset` function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to easily recreate target application parameter values from a number of application runs.

See Also

`xpctarget.xpc.loadparamset`

Purpose

Change target application object property values

Syntax

MATLAB command line

```
set(target_object)
set(target_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
target_object.set('property_name1', 'property_value1')
set(target_object, property_name_vector,
property_value_vector)
target_object.property_name = property_value
```

Target computer command line - Commands are limited to the target object properties `stoptime`, `sampletime`, and parameters.

```
parameter_name = parameter_value
stoptime = floating_point_number
sampletime = floating_point_number
```

Arguments

`target_object`	Name of a target object.
`'property_name'`	Name of a target object property. Always use quotation marks.
`property_value`	Value for a target object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

`set` sets the properties of the target object. Not all properties are user writable.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`. The writable properties for a target object

are listed in the following table. This table includes a description of the properties:

Property	Description	Writable
`CommunicationTimeOut`	Communication timeout between host and target computer, in seconds.	Yes
`LogMode`	Controls which data points are logged: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. See "User Interaction" for limitations on target property changes to sample times.	Yes

Property	Description	Writable
`ShowParameters`	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`ShowSignals`	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are `'on'` and `'off'`.	Yes
`StopTime`	Time when the target application stops running. Values are in seconds. The original value is set in the **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes
`ViewMode`	Display either all scopes or a single scope on the target computer. Value is `'all'` or a single scope index. This property is active only if the environment property `TargetScope` is set to `enabled`.	Yes

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the value of the properties after the indicated settings have been made.

Examples

Get a list of writable properties for a scope object.

```
set(tg)
ans =
          StopTime: {}
        SampleTime: {}
          ViewMode: {}
           LogMode: {}
    ShowParameters: {}
       ShowSignals: {}
```

Change the property `ShowSignals` to on.

```
tg.set('showsignals', 'on') or set(tg, 'showsignals', 'on')
```

As an alternative to the method `set`, use the target object property `ShowSignals`. In the MATLAB window, type

```
tg.showsignals ='on'
```

See Also

`get` | `set` | `xpctarget.xpc.get (target application object)` | `xpctarget.xpcsc.get (scope object)` | `xpctarget.xpcsc.set (scope object)`

How To

- "Application and Driver Scripts"

Purpose

Change writable target object parameters

Syntax

MATLAB command line

```
setparam(target_object, parameter_index, parameter_value)
```

Arguments

`target_object`	Name of an existing target object. The default name is `tg`.
`parameter_index`	Index number of the parameter.
`parameter_value`	Value for a target object parameter.

Description

Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- `parIndexVec`
- `OldValues`
- `NewValues`

Examples

Set the value of parameter index `5` to `100`.

```
setparam(tg, 5, 100)
ans =
parIndexVec: 5
OldValues: 400
NewValues: 100
```

Simultaneously set values for multiple parameters. Use the cell array format to specify new parameter values.

```
setparam(tg, [1 5],{10,100})
ans =
parIndexVec: [1 5]
OldValues: {[2]  [4]}
```

```
NewValues: {[10]  [100]}
```

Purpose

Start execution of target application on target computer

Syntax

MATLAB command line

```
start(target_object)
target_object.start
+target_object
```

Target computer command line

```
start
```

Arguments

`target_object`	Name of a target object. The default name is `tg`.

Description

Method of both target and scope objects. Starts execution of the target application represented by the target object. Before using this method, the target application must be created and loaded on the target computer. If a target application is running, this command has no effect.

Examples

Start the target application represented by the target object `tg`.

```
+tg
tg.start
start(tg)
```

See Also

`xpctarget.xpc.stop (target application object)` | `xpctarget.xpc.load` | `xpctarget.xpc.unload` | `xpctarget.xpcsc.stop (scope object)`

xpctarget.xpc.stop (target application object)

Purpose

Stop execution of target application on target computer

Syntax

MATLAB command line

```
stop(target_object)
target_object.stop
-target_object
```

Target computer command line

```
stop
```

Arguments

`target_object`	Name of a target object.

Description

Stops execution of the target application represented by the target object. If the target application is stopped, this command has no effect.

Examples

Stop the target application represented by the target object `tg`.

```
stop(tg) or tg.stop or -tg
```

See Also

`xpctarget.xpc.start (target application object)` | `xpctarget.xpcsc.stop (scope object)` | `xpctarget.xpcsc.start (scope object)`

Purpose

Test communication between host and target computers

Syntax

MATLAB command line

```
targetping(target_object)
target_object.targetping
```

Arguments

`target_object` Name of a target object.

Description

Method of a target object. Use this method to ping a target computer from the host computer. This method returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target, otherwise it returns `failed`.

This function works with both RS-232 and TCP/IP communication.

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

Examples

Ping the communication between the host and the target object `tg`.

```
targetping(tg) or tg.targetping
```

See Also

`xpctarget.xpc`

xpctarget.xpc.unload

Purpose

Remove current target application from target computer

Syntax

MATLAB command line

```
unload(target_object)
target_object.unload
```

Arguments

`target_object`	Name of a target object that represents a target application.

Description

Method of a target object. The kernel goes into loader mode and is ready to download new target application from the host computer.

Note If you are running in StandAlone mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

Examples

Unload the target application represented by the target object `tg`.

```
unload(tg) or tg.unload
```

See Also

`xpctarget.xpc.load` | `xpctarget.xpc.reboot`

Purpose

Control and access properties of file scopes

Description

The scope gets a data package from the kernel and stores the data in a file in the target computer file system. Depending on the setting of `WriteMode`, the file size is or is not continuously updated. You can then transfer the data to another computer for examination or plotting.

Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get (scope object)`	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set (scope object)`	Change property values for scope objects
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

These properties are specific to class `xpcfs`.

Property	Description	Writeable
`AutoRestart`	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data. To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	No

Property	Description	Writeable
	For host or target scopes, this parameter has no effect.	
`DynamicFileName`	Values are `'on'` and `'off'`. By default, the value is `'off'`. Enable the ability to dynamically create multiple log files for file scopes. To use `DynamicFileName`, set `AutoRestart` to `'on'` first. When you enable `DynamicFileName`, configure `Filename` to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope. You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters. For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writeable

`Filename`	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
		No

Property	Description	Writeable
	The last file name of this series will be `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
`MaxWriteFileSize`	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file. For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).	Yes

Property	Description	Writeable
	For host or target scopes, this parameter has no effect.	
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	Yes

xpctarget.xpcsc.addsignal

Purpose

Add signals to scope represented by scope object

Syntax

MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

Target command line

```
addsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object_vector`	Name of a single scope object or the name of a vector of scope objects.
`signal_index_vector`	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
`scope_index`	Single scope index.

Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

Note You must stop the scope before you can add a signal to it.

Examples

Add signals `0` and `1` from the target object `tg` to the scope object `sc1`. The signals are added to the scope, and the scope object property `Signals` is updated to include the added signals.

```
sc1 = getscope(tg,1)
addsignal(sc1,[0,1]) or sc1.addsignal([0,1])
```

Display a list of properties and values for the scope object `sc1` with the property `Signals`, as shown below.

```
sc1.Signals
Signals              = 1  : Signal Generator
                       0  : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1]
```

Or, to directly assign signal values to the scope object property `Signals`,

```
sc1.signals = [0,1]
```

See Also

`xpctarget.xpcsc.remsignal` | `xpctarget.xpcsc.set (scope object)` | `xpctarget.xpc.addscope` | `xpctarget.xpc.getsignalid`

xpctarget.xpcsc.get (scope object)

Purpose

Return property values for scope objects

Syntax

MATLAB command line

```
get(scope_object_vector)
get(scope_object_vector, 'scope_object_property')
get(scope_object_vector, scope_object_property_vector)
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope or name of a vector of scope objects.
`scope_object_property`	Name of a scope object property.

Description

`get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`AutoRestart`	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop.	No

Property	Description	Writable
	If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data. For host or target scopes, this parameter has no effect. To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	
`Data`	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`DisplayMode`	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For host or file scopes, this parameter has no effect. .	Yes
`DynamicFileName`	Values are `'on'` and `'off'`. By default, the value is `'off'`. Enable the ability to dynamically create multiple log files for file scopes. To use `DynamicFileName`, set `AutoRestart` to `'on'` first. When you enable `DynamicFileName`, configure `Filename` to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.	Yes

Property	Description	Writable
	You can enable the creation of up to 99999999 files (<%%%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters. For host or file scopes, this parameter has no effect.	
Filename	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to FileName. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope. If you set DynamicFileName and AutoRestart to 'on', configure Filename to dynamically increment. Use a base file name, an underscore (_), and a < > specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for Filename, C:\work\file_<%%%>.dat creates file names with the following pattern: file_001.dat file_002.dat file_003.dat The last file name of this series will be file_999.dat. If the function is still logging data when the last file name reaches its maximum	No

Property	Description	Writable
	size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost. For host or target scopes, this parameter has no effect.	
`MaxWriteFileSize`	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file.	Yes
`Grid`	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes

Property	Description	Writable
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`Time`	Contains the time data for a single data package from a scope.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). For host or target scopes, this parameter has no effect.	Yes

Property	Description	Writable
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For host or file scopes, this parameter has no effect.	Yes

Examples

List all the readable properties, along with their current values. This is given in the form of a structure whose field names are the property names and whose field values are property values.

```
get(sc)
```

List the value for the scope object property `Type`. Notice that the property name is a string, in quotation marks, and is not case sensitive.

```
get(sc,'type')
ans = Target
```

See Also

`get` | `set` | `xpctarget.xpcsc.set (scope object)` | `xpctarget.xpc.set (target application object)`

xpctarget.xpcsc Class

Purpose

Base class for all scope classes

Description

This is the base class for the scope classes, `xpctarget.xpcfs Class`, `xpctarget.xpcschost Class`, and `xpctarget.xpcsctg Class`. All methods and properties are inherited by the derived classes. When a mixture of derived classes are stored in a scope collection, only the base class methods and properties are available. All scope class constructors are `Private` and are not intended to be called from the MATLAB prompt.

A scope acquires data from the target application and displays that data on the target computer, uploads the data to the host computer, or stores that data in a file in the target computer file system. All target, host, or file scopes run on the target computer.

Methods

These methods are inherited by the derived classes.

Method	Description
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get (scope object)`	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set (scope object)`	Change property values for scope objects
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software trigger start of data acquisition for scope(s)

Properties

These properties are inherited by the derived classes.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes

Property	Description	Writable
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

xpctarget.xpcsc.remsignal

Purpose

Remove signals from scope represented by scope object

Syntax

MATLAB command line

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

Target command line

```
remsignal scope_index = signal_index, signal_index, . . .
```

Arguments

`scope_object`	MATLAB object created with the target object method `addscope` or `getscope`.
`signal_index_vector`	Index numbers from the scope object property `Signals`. This argument is optional, and if it is left out all signals are removed.
`signal_index`	Single signal index.

Description

`remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

Note You must stop the scope before you can remove a signal from it.

Examples

Remove signals 0 and 1 from the scope represented by the scope object sc1.

```
sc1.get('signals')
ans= 0 1
```

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

```
remsignal(sc1,[0,1])
```

or

```
sc1.remsignal([0,1])
```

See Also

`xpctarget.xpcsc.remsignal` | `xpctarget.xpc.getsignalid`

xpctarget.xpcsc.set (scope object)

Purpose

Change property values for scope objects

Syntax

MATLAB command line

```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

Arguments

`scope_object`	Name of a scope object or a vector of scope objects.
`'property_name'`	Name of a scope object property. Always use quotation marks.
`property_value`	Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user writable. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the values of the properties after the indicated settings have been made.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`AutoRestart`	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data. For host or target scopes, this parameter has no effect. To use the `DynamicFileName` property, set `AutoRestart` to `'on'` first.	No
`Data`	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`DisplayMode`	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For host or file scopes, this parameter has no effect. .	Yes
`DynamicFileName`	Values are `'on'` and `'off'`. By default, the value is `'off'`. Enable the ability to dynamically create multiple log files for file scopes. To use `DynamicFileName`, set `AutoRestart` to `'on'` first. When you enable `DynamicFileName`, configure `Filename` to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope. You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters. For host or file scopes, this parameter has no effect.	Yes

Property	Description	Writable

`Filename`	Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat` for scope blocks. Note that for file scopes created through the MATLAB interface, there is no name initially assigned to `FileName`. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.
	
	If you set `DynamicFileName` and `AutoRestart` to `'on'`, configure `Filename` to dynamically increment. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for `Filename`, `C:\work\file_<%%%>.dat` creates file names with the following pattern:
	
	`file_001.dat`
	`file_002.dat`
	`file_003.dat`
	
	The last file name of this series will be `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.
	
	For host or target scopes, this parameter has no effect.
		No

Property	Description	Writable
`MaxWriteFileSize`	Provide the maximum size of `Filename`, in bytes. This value must be a multiple of `WriteSize`. Default is `536870912`. When the size of a log file reaches `MaxWriteFileSize`, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create any additional log files, it overwrites the first log file.	Yes
`Grid`	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes

Property	Description	Writable
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No
`Time`	Contains the time data for a single data package from a scope.	No
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes

Property	Description	Writable
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes

Property	Description	Writable
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always knows the actual file size. With `'Lazy'` mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact). For host or target scopes, this parameter has no effect. .	Yes
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides optimal performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`.	Yes

Property	Description	Writable
	For host or file scopes, this parameter has no effect.	

Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
           NumSamples: {}
           Decimation: {}
          TriggerMode: {5x1 cell}
        TriggerSignal: {}
         TriggerLevel: {}
         TriggerSlope: {4x1 cell}
         TriggerScope: {}
        TriggerSample: {}
              Signals: {}
    NumPrePostSamples: {}
                 Mode: {5x1 cell}
               YLimit: {}
                 Grid: {}
```

The property value for the scope object `sc1` is changed to `on`:

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

See Also

`get | set | xpctarget.xpcsc.get (scope object) | xpctarget.xpc.set (target application object) | xpctarget.xpc.get (target application object)`

Purpose

Start execution of scope on target computer

Syntax

MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector))
```

Target computer command line

```
startscope scope_index
startscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add xPC Target scope blocks to your Simulink model.

Examples

Start one scope with the scope object sc1.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
start(sc1) or sc1.start or +sc1
```

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes=
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2])
```

Start all scopes:

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

```
start(getscope(tg)) or start(tg.getscope)
```

See Also

xpctarget.xpc.getscope | xpctarget.xpc.stop (target application object) | xpctarget.xpcsc.stop (scope object)

Purpose

Stop execution of scope on target computer

Syntax

MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

Target computer command line

```
stopscope scope_index
stopscope 'all'
```

Arguments

`target_object`	Name of a target object.
`scope_object_vector`	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

Method for scope objects. Stops the scopes represented by the scope objects.

Examples

Stop one scope represented by the scope object `sc1`.

```
stop(sc1) or sc1.stop or -sc1
```

Stop all scopes with a scope object vector `allscopes` created with the command

```
allscopes = getscope(tg) or allscopes = tg.getscope.
stop(allscopes) or allscopes.stop or -allscopes
```

or type

```
stop(getscope(tg)) or stop(tg.getscope)
```

See Also

xpctarget.xpc.getscope | xpctarget.xpc.stop (target application object) | xpctarget.xpc.start (target application object) | xpctarget.xpcsc.start (scope object)

Purpose

Software-trigger start of data acquisition for scope(s)

Syntax

MATLAB command line

```
trigger(scope_object_vector) or scope_object_vector.trigger
```

Arguments

`scope_object_vector`	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form `[scope_object1, scope_object2]`, or the target object method `getscope`, which returns a `scope_object` vector.

Description

Method for a scope object. If the scope object property `TriggerMode` has a value of `'software'`, this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property `NumSamples`.

Note that only scopes with type `host` store data in the properties `scope_object.Time` and `scope_object.Data`.

Examples

Set a single scope to software trigger, trigger the acquisition of one set of samples, and plot data.

```
sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1)
sc1.triggermode = 'software'
tg.start, or start(tg), or +tg
sc1.start  or start(sc1) or +sc1
sc1.trigger or trigger(sc1)
plot(sc1.time, sc1.data)
sc1.stop or stop(sc1) or -sc1
tg.stop or stop(tg) or -tg1
```

Set all scopes to software trigger and trigger to start.

```
allscopes = tg.getscopes
```

```
allscopes.triggermode = 'software'
allscopes.start or start(allscopes) or +allscopes
allscopes.trigger or trigger(allscopes)
```

Purpose

Control and access properties of host scopes

Description

The scope gets a data package from the kernel, waits for an upload command from the host computer, and uploads the data to the host. The host computer displays the data using a scope viewer or other MATLAB functions.

Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get (scope object)`	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set (scope object)`	Change property values for scope objects
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcschost.

Property	Description	Writeable
Data	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
Time	Contains the time data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No

Purpose

Control and access properties of target scopes

Description

The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of `DisplayMode`, the data may be displayed numerically or graphically by a redrawing, sliding, and rolling display.

Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
`xpctarget.xpcsc.addsignal`	Add signals to scope represented by scope object
`xpctarget.xpcsc.get (scope object)`	Return property values for scope objects
`xpctarget.xpcsc.remsignal`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.set (scope object)`	Change property values for scope objects
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software-trigger start of data acquisition for scope(s)

Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No
`Decimation`	A number `n`, where every `n`th sample is acquired in a scope window.	Yes

Property	Description	Writable
`NumPrePostSamples`	For host or target scopes, this parameter is the number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes
`NumSamples`	Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes. For file scopes, this parameter works in conjunction with the **AutoRestart** check box. If the **AutoRestart** box is selected, the file scope collects data up to **Number of Samples**, then starts over again, overwriting the buffer. If the **AutoRestart** box is not selected, the file scope collects data only up to **Number of Samples**, then stops.	Yes
`ScopeId`	A numeric index, unique for each scope.	No
`Signals`	List of signal indices from the target object to display on the scope.	Yes
`Status`	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are `'Acquiring'`, `'Ready for being Triggered'`, `'Interrupted'`, and `'Finished'`.	No

Property	Description	Writable
`TriggerLevel`	If `TriggerMode` is `'Signal'`, indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
`TriggerMode`	Trigger mode for a scope. Valid values are `'FreeRun'` (default), `'Software'`, `'Signal'`, and `'Scope'`.	Yes
`TriggerSample`	If `TriggerMode` is `'Scope'`, then `TriggerSample` specifies which sample of the triggering scope the current scope should trigger on. For example, if `TriggerSample` is `0` (default), the current scope triggers on sample `0` (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If `TriggerSample` is `1`, the first sample (sample `0`) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope. As a special case, setting `TriggerSample` to `-1` means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.	Yes
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes

Property	Description	Writable
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the host computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`.	Yes

These properties are specific to class `xpcsctg`.

Property	Description	Writeable
`DisplayMode`	For target scopes, indicate how a scope displays the signals. Values are `'Numerical'`, `'Redraw'` (default), `'Sliding'`, and `'Rolling'`. For host or file scopes, this parameter has no effect. .	Yes
`Grid`	Values are `'on'` and `'off'`. For host or file scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`Mode`	**Note** The `Mode` property will be removed in a future release. • For target scopes, use `DisplayMode`. • For file scopes, use `WriteMode`. • For host scopes, this parameter has no effect.	Yes
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For host or file scopes, this parameter has no effect.	Yes

xpctargetping

Purpose

Test communication between host and target computers

Syntax

MATLAB command line

```
xpctargetping
xpctargetping('mode', 'arg1', 'arg2')
```

Arguments

`mode` — Optionally, enter the communication mode:

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

`TCPIP`	Enable TCP/IP connection with target computer.
`RS232`	Enable RS-232 connection with target computer.

`arg1` — Optionally, enter an argument based on the `mode` value:

IP address	If `mode` is `'TCPIP'`, enter the IP address of the target computer.
COM port	If `mode` is `'RS232'`, enter the host COM port.

`arg2` — Optionally, enter an argument based on the `mode` value:

Port	If `mode` is `'TCPIP'`, enter the port number for the target computer.
Baud rate	If `mode` is `'RS232'`, enter the baud rate for the connection between the host and target computer.

Description

Pings the target computer from the host computer and returns either `success` or `failed`. If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetping
```

If you have multiple target computers in your system, use the following syntax to identify the target computer to ping.

```
xpctargetping('mode', 'arg1', 'arg2')
```

This function returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target.

This function works with both RS-232 and TCP/IP communication.

```
ans =
success
```

Examples

Check for communication between the host computer and target computer.

```
xpctargetping
```

If you have a serial connection with the target computer you want to check, use the following syntax.

```
xpctargetping('RS232', 'COM1', '115200')
```

How To

- "Run Confidence Test on Configuration"

xpctargetspy

Purpose

Open Real-Time xPC Target Spy window on host computer

Syntax

MATLAB command line

```
xpctargetspy
xpctargetspy(target_object)
xpctargetspy('target_object_name')
```

Arguments

`target_object`	Variable name to reference the target object.
`target_object_name`	Target object name as specified in the xPC Target Explorer.

Description

This graphical user interface (GUI) allows you to upload displayed data from the target computer. By default, `xpctargetspy` opens a Real-Time xPC Target Spy window for the target object, `tg`. If you have multiple target computers in your system, you can call the `xpctargetspy` function for a particular target object, `target_object`.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetspy
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object first.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

Then, use the following syntax.

```
xpctargetspy(target_object)
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax.

```
target_object=xpctarget.xpc('target_object_name')
```

The behavior of this function depends on the value for the environment property `TargetScope`:

- If `TargetScope` is enabled, a single graphics screen is uploaded. The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode. You must explicitly request an update. To manually update the host screen with another target screen, move the pointer into the Real-Time xPC Target Spy window and right-click to select **Update xPC Target Spy**.
- If `TargetScope` is disabled, text output is transferred once every second to the host and displayed in the window.

Examples

To open the Real-Time xPC Target Spy window for a default target computer, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the Real-Time xPC Target Spy window for a target computer, tg1, in the MATLAB window, type

```
xpctargetspy(tg1)
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object, tg2, first.

```
tg2=xpctarget.xpc('RS232', 'COM1', '115200')
```

Then, use the following syntax.

```
xpctargetspy(tg2)
```

Purpose

Test xPC Target installation

Syntax

MATLAB command line

```
xpctest
xpctest('target_name')
xpctest('-noreboot')
xpctest('noreboot')
xpctest('target_name', 'noreboot')
xpctest('target_name', '-noreboot')
```

Arguments

`'target_name'`	Name of target computer to test.
`'noreboot'`	Only one possible option. Skips the reboot test. Use this option if the target hardware does not support software rebooting. Value is `'noreboot'` or `'-noreboot'`.

Description

`xpctest` is a series of xPC Target tests to check the functioning of the following xPC Target tasks:

- Initiate communication between the host and target computers.
- Reboot the target computer to reset the target environment.
- Build a target application on the host computer.
- Download a target application to the target computer.
- Check communication between the host and target computers using commands.
- Execute a target application.
- Compare the results of a simulation and the target application run.

`xpctest('noreboot')` or `xpctest('-noreboot')` skips the reboot test on the default target computer. Use this option if target hardware does not support software rebooting.

`xpctest('target_name')` runs the tests on the target computer identified by `'target_name'`.

`xpctest('target_name', 'reboot')` or `xpctest('target_name', '-reboot')` runs the tests on the target computer identified by `'target_name'`, but skips the reboot test.

Examples

If the target hardware does not support software rebooting, or to skip the reboot test, in the MATLAB window, type

```
xpctest('-noreboot')
```

To run `xpctest` on a specified target computer, for example TargetPC1, type

```
xpctest('TargetPC1')
```

How To

- "Run Confidence Test on Configuration"
- "Test 1: Ping Using System Ping"

xpcwwwenable

Purpose

Disconnect target computer from current client application

Syntax

MATLAB command line

```
xpcwwwenable
xpcwwwenable('target_obj_name')
```

Description

Use this function to disconnect the target application from the MATLAB interface before you connect to the Web browser. You can also use this function to connect to the MATLAB interface after using a Web browser, or to switch to another Web browser.

`xpcwwwenable('target_obj_name')` disconnects the target application on `target_obj_name`, for example `'TargetPC1'`, from the MATLAB interface.

3

xPC Target API Reference for C

C API Functions

In this section...

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
xPCFSGetError	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
EINVSCTYPE	Invalid Scope type
EINVSIGIDX	Invalid Signal index
EINVTRIGMODE	Invalid trigger mode
EINVTRIGSLOPE	Invalid Trigger Slope Value
EINVTRSCIDX	Invalid Trigger Scope index
EINVNUMSAMP	Number of samples must be nonnegative
EINVSTARTVAL	Invalid value for "start"
EINVTFIN	Invalid value for TFinal
EINVTS	Invalid value for Ts (must be between 8e-6 and 10)
EINVWSVER	Invalid Winsock version (1.1 needed)
EINVXPCVERSION	Target has an invalid version of 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

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

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

Description

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

See Also

API function xPCFSDirItems

Purpose

Type definition for file system disk information structure

Syntax

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

Fields

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

FATType	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
FATCount	This value contains the number of FAT partitions on the volume.
MaxDirEntries	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
BytesPerSector	This value contains the sector size. This value is most likely to be `512`.
SectorsPerCluster	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
TotalClusters	This value contains the number of file storage clusters on the volume.
BadClusters	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
FreeClusters	This value contains the number of clusters that are currently available for storage.
Files	This value contains the number of files, including directories, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
FileChains	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of `Files`.

FreeChains	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
LargestFreeChain	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to `FreeClusters`.

Description

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

See Also

API function `xPCFSDiskInfo`

fileinfo

Purpose

Type definition for file information structure

Syntax

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

Fields

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

Description

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

See Also

`xPCFSFileInfo`

Purpose

Type definition for logging options structure

Syntax

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

Fields

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

Description

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

See Also

API functions xPCSetLogMode, xPCGetLogMode

scopedata

Purpose

Type definition for scope data structure

Syntax

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

Fields

number — The scope number.

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

1	Host
2	Target

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

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

	`5` Scope is preacquiring data
`signals`	List of signal indices from the target object to display on the scope.
`numsamples`	Number of contiguous samples captured during the acquisition of a data package.
`decimation`	A number, `N`, meaning every `N`th sample is acquired in a scope window.
`triggermode`	Trigger mode for a scope. Values are one of the following: `0` FreeRun (default) `1` Software `2` Signal `3` Scope
`numprepostsamples`	If this value is less than `0`, this is the number of samples to be saved before a trigger event. If this value is greater than `0`, this is the number of samples to skip after the trigger event before data acquisition begins.
`triggersignal`	If *`triggermode`* is `2` (Signal), identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.
`triggerscope`	If *`triggermode`* is `3` (Scope), identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.
`triggerscopesample`	If *`triggermode`* is `3` (Scope), specifies the number of samples to be acquired by the triggering scope before triggering a second scope. This must be a nonnegative value.

triggerlevel	If *triggermode* is 2 (Signal), indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with either a rising or falling signal.
triggerslope	If *triggermode* is 2 (Signal), indicates whether the trigger is on a rising or falling signal. Values are: 0 Either rising or falling (default) 1 Rising 2 Falling

Description

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

See Also

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

Purpose

Create new scope

Prototype

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

Arguments

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

Description

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

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

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

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

See Also

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

Target object method `xpctarget.xpc.addscope`

xPCAverageTET

Purpose

Return average task execution time

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API functions `xPCMaximumTET`, `xPCMinimumTET`

Target object property `AvgTET`

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 effect 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);
```

Arguments

none

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

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

Description

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

See Also

API function `xPCFSGetPWD`

File object method `xpctarget.fsbase.cd`

Purpose

Close file on target computer

Prototype

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

Arguments

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

Description

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

See Also

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

File object method `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

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

Description

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

See Also

API function `xPCFSDirSize`

File object method `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.fsbase.dir

xPCFSDirSize

Purpose

Return size of specified folder listing on target computer

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCFSDirItems`

File object method `xpctarget.fsbase.dir`

Purpose

Get number of items in folder

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCFSDir`

File object method `xpctarget.fsbase.dir`

xPCFSDiskInfo

Purpose

Information about target computer file system

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
driveletter	Enter the drive letter of the file system for which you want information.

Description

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

See Also

API structure `xpctarget.fs.diskinfo`

Purpose

Return information for open file on target computer

Prototype

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

Arguments

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

Description

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

See Also

Structure `xpctarget.fs.fileinfo`

xPCFSGetError

Purpose

Get text description for error number on target computer file system

Prototype

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

Arguments

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

Description

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

Purpose

Return size of file on target computer

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API functions `xPCFSOpenFile`, `xPCFSReadFile`

File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

xPCFSGetPWD

Purpose

Get current folder of target computer

Prototype

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

Arguments

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

Description

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

See Also

File object method `xpctarget.fsbase.pwd`

Purpose

Create new folder on target computer

Prototype

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

Arguments

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

Description

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

See Also

API function `xPCFSGetPWD`

File object method `xpctarget.fsbase.mkdir`

xPCFSOpenFile

Purpose

Open file on target computer

Prototype

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

Arguments

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

Return

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

Description

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

See Also

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

File object methods `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

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

Description

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

See Also

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

File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

xPCFSRemoveFile

Purpose

Remove file from target computer

Prototype

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

Arguments

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

Description

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

See Also

File object method `xpctarget.fs.removefile`

Purpose

Remove folder from target computer

Prototype

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

Arguments

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

Description

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

See Also

File object method `xpctarget.fsbase.rmdir`

xPCFSScGetFilename

Purpose

Get name of file for scope

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCFSScSetFilename`

Scope object property `Filename`

Purpose

Get write mode of file for scope

Prototype

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

Arguments

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

Return

Returns the number indicating the write mode. Values are

0	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
1	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always has 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

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

Return

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

Description

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

See Also

API function `xPCFSScSetWriteSize`

Scope object property `WriteSize`

Purpose

Specify name for file to contain signal data

Prototype

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

Arguments

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

Description

The `xPCFSScSetFilename` function sets the name of the file to which the scope will save the signal data. The 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

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

Description

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

`0`	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
`1`	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always has 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

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

Description

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

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

Description

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

See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSReadFile`

Purpose

Get version number of xPC Target API

Prototype

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

Arguments

none

Return

The `xPCGetApiVersion` function returns a string with the version number of the xPC Target kernel on the target computer.

Description

The `xPCGetApiVersion` 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 `xPCGetTargetVersion`

xPCGetAppName

Purpose

Return target application name

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
model_name	The `xPCGetAppName` function copies the target application name string into the buffer pointed to by *model_name*. *model_name* is then returned. You can later use *model_name* in a function such as `printf`. Note that the maximum size of the buffer is 256 bytes. To reserve enough space for the application name string, allocate a buffer of size 256 bytes.

Return

The `xPCGetAppName` function returns a string with the name of the target application.

Description

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

Examples

Allocate 256 bytes for the buffer `appname`.

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

See Also

API function `xPCIsAppRunning`

Target object property `Application`

Purpose

Return display mode for target message window

Prototype

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

Arguments

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

Return

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

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

Return

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

Description

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

See Also

API function `xPCSetEcho`

xPCGetExecTime

Purpose

Return target application execution time

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API functions `xPCSetStopTime`, `xPCGetStopTime`

Target object property `ExecTime`

Purpose

Return constant of last error

Prototype

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

Return

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

Description

The `xPCGetLastError` function returns the constant of the last reported error by another API function. This value is reset every time you call a new function. Therefore, you should check this constant value immediately after a call to an API function. For a list of error constants and messages, see "C API Error Messages" on page 3-10.

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

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCSetLogMode`

API structure `lgmode`

Purpose

Return number of outputs

Prototype

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

Arguments

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

Return

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

Description

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

See Also

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

xPCGetNumParams

Purpose

Return number of tunable parameters

Prototype

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

Arguments

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

Return

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

Description

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

See Also

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

Target object property `NumParameters`

Purpose

Return number of scopes added to target application

Prototype

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

Arguments

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

Return

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

Description

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

xPCGetNumScSignals

Purpose

Returns number of signals added to specific scope

Prototype

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

Arguments

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

Return

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

Description

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

Purpose

Return number of signals

Prototype

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

Arguments

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

Return

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

Description

The `xPCGetNumSignals` function returns the total number of signals in the target application that can be monitored from the host. Use this function to see how many signals you can monitor.

See Also

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

Target object property `NumSignals`

xPCGetNumStates

Purpose

Return number of states

Prototype

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

Arguments

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

Return

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

Description

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

See Also

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

Target object property `StateLog`

Purpose

Copy output log data to array

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
first_sample	Enter the index of the first sample to copy.
num_samples	Enter the number of samples to copy from the output log.
decimation	Select whether to copy all the sample values or every `N`th value.
output_id	Enter an output identification number.
output_data	The log is stored in *output_data*, whose allocation is the responsibility of the caller.

Description

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

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

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

See Also

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

Target object method `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

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

Description

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

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

See Also

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

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

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

Description

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

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

See Also

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

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

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

Return

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

Description

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

See Also

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

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

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
paramIdx	Enter a parameter index.
blockName	String with the full block path generated by Simulink Coder.
paramName	Name of a parameter for a specific block.

Description

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

See Also

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

Target object properties `ShowParameters`, `Parameters`

Purpose

Return target application sample time

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCSetSampleTime`

Target object property `SampleTime`

xPCGetScope

Purpose

Get and copy scope data to structure

Prototype

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

Arguments

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

Return

The `xPCGetScope` function returns a structure of type `scopedata`.

Description

The `xPCGetScope` function gets properties of a scope with *scNum* and copies the properties into a structure with type `scopedata`. You can use this function in conjunction with `xPCSetScope` to change several properties of a scope at one time. See `scopedata` for a list of properties. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCSetScope`, `scopedata`

Target object method `xpctarget.xpc.getscope`

Purpose

Get and copy list of scope numbers

Prototype

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

Arguments

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

Description

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

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

xPCGetScopes

Purpose

Get and copy list of scope numbers

Prototype

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

Arguments

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

Description

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

Note This function will be 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

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

Arguments

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

Return

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

Description

The `xPCGetSessionTime` function returns, as a double, the amount of time in seconds that the 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

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigNum	Enter a signal number.

Return

The `xPCGetSignal` function returns the current value of signal *sigNum*. If the function detects an error, it returns `-1`.

Description

The `xPCGetSignal` function returns the current value of a signal. For vector signals, use `xPCGetSignals` rather than call this function multiple times. Use the `xPCGetSignalIdx` function to get the signal number.

See Also

API function `xPCGetSignals`

Target object properties `ShowSignals`, `Signals`

Purpose

Return index for signal

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigName	Enter a signal name.

Return

The `xPCGetSignalIdx` function returns the index for the signal with name *sigName*. If the function detects an error, it returns `-1`.

Description

The `xPCGetSignalIdx` function returns the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file `model_namebio.m` in the generated code, where *model_name* is the name of the model. The creator of the application should already know the signal name.

See Also

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

Target object method `xpctarget.xpc.getsignalid`

xPCGetSigIdxfromLabel

Purpose

Return array of signal indices

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigLabel	String with the name of a signal label.
sigIds	Return array of signal indices.

Return

If `xPCGetSigIdxfromLabel` finds a signal, it fills an array *sigIds* with signal indices and returns `0`. If it finds no signal, it returns `-1`.

Description

The `xPCGetSigIdxfromLabel` function returns in *sigIds* the array of signal indices for signal *sigName*. This function assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the 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

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

Return

The xPCGetSignalLabel function returns the label of the signal.

Description

The xPCGetSignalLabel function copies and returns the signal label, including the block path, of a signal with *sigIdx*. The result is stored in *sigLabel*. If *sigIdx* is invalid, xPCGetLastError returns a nonzero value, and *sigLabel* is unchanged. The function returns *sigLabel*, which makes it convenient to use in a printf or similar statement. This function assumes that you already know the signal index. Signal labels must be unique.

This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the 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

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigName	String with the name of a signal.

Return

The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. If the function detects an error, it returns `-1`.

Description

The `xPCGetSigLabelWidth` function returns the number of elements that the signal *`sigName`* contains. This function assumes that you have labeled the signal for which you request the elements (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the 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

port	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
sigIdx	Enter a signal index.
sigName	String with the name of a signal.

Return

The xPCGetSignalName function returns the name of the signal.

Description

The xPCGetSignalName function copies and returns the signal name, including the block path, of a signal with *sigIdx*. The result is stored in *sigName*. If *sigIdx* is invalid, xPCGetLastError returns a nonzero value, and *sigName* is unchanged. The function returns *sigName*, which makes it convenient to use in a printf or similar statement. This function assumes that you already know the signal index.

See Also

API functions xPCGetSignalIdx, xPCGetSignalWidth, xPCGetSignal, xPCGetSignals

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

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

Return

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

Description

The `xPCGetSignals` function is the vector version of the function `xPCGetSignal`. This function returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step (for that, define a scope of type `SCTYPE_HOST` and use `xPCScGetData`). `xPCGetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same effect. However, the `xPCGetSignals` function is faster, and the signal values are more likely to be spaced closely together. The signals are converted to doubles regardless of the actual data type of the signal.

For *signals*, the list you provide should be stored in an integer array. Get the signal numbers with the function `xPCGetSignalIdx`.

See Also

API function `xPCGetSignal`, `xPCGetSignalIdx`

Example

To reference signal vector data rather than scalar values, pass a vector of indices for the signal data. For example:

```
/*********************************************************/

/* Assume a signal of width 10, with the blockpath
* mySubsys/mySignal and the signal index s1.
*/

int i;
int sigId[10];
double sigVal[10]; /* Signal values are stored here */

/* Get the ID of the first signal */
sigId[0] = xPCGetSignalIdx(port, "mySubsys/mySignal/s1");

if (sigId[0] == -1) {
/* Handle error */
}

for (i = 1; i < 10; i++) {
     sigId[i] = sigId[0] + i;
}

xPCGetSignals(port, 10, sigId, sigVal);
/* If no error, sigVal should have the signal values */

/*********************************************************/
```

To repeatedly get the signals, repeat the call to `xPCGetSignals`. If you do not change `sigID`, you only need to call `xPCGetSignalIdx` once.

xPCGetSignalWidth

Purpose

Return width of signal

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
sigIdx	Enter the index of a signal.

Return

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

Description

The `xPCGetSignalWidth` function returns the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector again. A signal's width is the number of signals in the vector.

See Also

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

Purpose

Copy state log values to array

Prototype

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

Arguments

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

Description

The xPCGetStateLog function gets the state log. It then copies the log into *state_data*. You get the data for each state signal in turn by specifying the *state_id*. State IDs range from 1 to (N-1), where N is the return value of xPCGetNumStates. Entering 1 for *decimation* copies all values. Entering N copies every 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`, `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

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

Description

The `xPCGetTargetVersion` function gets a string with the version number of the 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

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

Description

The `xPCGetTETLog` function gets the task execution time (TET) log. It then copies the log into *TET_data*. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

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

See Also

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

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

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

Description

The `xPCGetTimeLog` function gets the time log and copies the log into *time_data*. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering `1` for *decimation* copies all values. Entering `N` copies every `N`th value. For *first_sample*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

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

See Also

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

Target object method `xpctarget.xpc.getlog`

Target object property `TimeLog`

Purpose

Initialize xPC Target DLL

Prototype

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

Arguments

none

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 `xpcinitfree.c`. Link this file with your application.

See Also

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

xPCIsAppRunning

Purpose

Return target application running status

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCIsOverloaded`

Target object property `Status`

Purpose

Return target computer overload status

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCIsAppRunning`

Target object property `CPUoverload`

xPCIsScFinished

Purpose

Return data acquisition status for scope

Prototype

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

Arguments

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

Return

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

Description

The `xPCIsScFinished` function returns a Boolean value depending on whether scope *`scNum`* is finished (state of `SCST_FINISHED`) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCScGetState`

Scope object property `Status`

Purpose

Load target application onto target computer

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
pathstr	Enter the full path to the target application file, excluding the file name. For example, in C, use a string like `"C:\\work"`.
filename	Enter the name of a compiled target application (`*.dlm`) without the file extension. For example, in C use a string like `"xpcosc"`.

Description

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

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

See Also

API functions `xPCStartApp`, `xPCStopApp`, `xPCUnloadApp`, `xPCSetLoadTimeOut`, `xPCGetLoadTimeOut`

Target object method `xpctarget.xpc.load`

Purpose

Restore parameter values

Prototype

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

Arguments

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

Description

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

See Also

API function `xPCSaveParamSet`

xPCMaxLogSamples

Purpose

Return maximum number of samples that can be in log buffer

Prototype

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

Arguments

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

Return

The `xPCMaxLogSamples` function returns the total number of samples. If the function detects an error, it returns `-1`.

Description

The `xPCMaxLogSamples` function returns the total number of samples that can be returned in the logging buffers.

See Also

API functions `xPCNumLogSamples`, `xPCNumLogWraps`, `xPCGetStateLog`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Target object property `MaxLogSamples`

Purpose

Copy maximum task execution time to array

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCMinimumTET`, `xPCAverageTET`

Target object property `MaxTET`

xPCMinimumTET

Purpose

Copy minimum task execution time to array

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCMaximumTET`, `xPCAverageTET`

Target object property `MinTET`

Purpose

Return number of samples in log buffer

Prototype

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

Arguments

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

Return

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

Description

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

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

See Also

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

xPCNumLogWraps

Purpose

Return number of times log buffer wraps

Prototype

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

Arguments

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

Return

The `xPCNumLogWraps` function returns the number of times the log buffer wraps. If the function detects an error, it returns `-1`.

Description

The `xPCNumLogWraps` function returns the number of times the log buffer wraps.

See Also

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

Target object property `NumLogWraps`

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

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

Return

The `xPCOpenSerialPort` function returns the port value for the connection. If the function detects an error, it returns `-1`.

Description

The `xPCOpenSerialPort` function initiates an RS-232 connection to an 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

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

Arguments

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

Return

The `xPCOpenTcpIpPort` function returns a nonnegative integer that you can then use as the port value for any 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 *`ipPort`* 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

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

Arguments

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

Return

The xPCReOpenPort function returns 0 if it reopens a connection without detecting an error. If the function detects an error, it returns -1.

Description

The xPCReOpenPort function reopens the communications channel pointed to by *port*. The difference between this function and xPCOpenSerialPort or xPCOpenTcpIpPort is that xPCReOpenPort uses the already existing settings, while the other functions need to set up the port.

See Also

API functions xPCOpenTcpIpPort, xPCClosePort

xPCRegisterTarget

Purpose

Register target with 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_TCPIP and COMMTYP_RS232 for *commType*. If *commType* is set to COMMTYP_RS232, the function ignores *ipAddress*

and *ipPort*. Analogously, the function ignores *comPort* and *baudRate* if *commType* is set to COMMTYP_TCPIP.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

See Also

API functions xPCDeRegisterTarget, xPCOpenTcpIpPort, xPCOpenSerialPort, xPCClosePort, xPCReOpenPort, xPCOpenConnection, xPCCloseConnection, xPCTargetPing

xPCRemScope

Purpose

Remove scope

Prototype

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

Arguments

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

Description

The `xPCRemScope` function removes the scope with number *scNum*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, see `xPCGetScopes`. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCAddScope`, `xPCScRemSignal`, `xPCGetScopes`

Target object method `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

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

Description

The `xPCScAddSignal` function adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScGetSignals` to get a list of the signals already present. Use the function `xPCGetScope` to get the scope number. Use the `xPCGetSignalIdx` function to get the signal number.

See Also

API functions `xPCScRemSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`

Scope object method `xpctarget.xpcsc.addsignal`

Purpose

Scope autorestart status

Prototype

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

Arguments

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

Return

The `xPCScGetAutoRestart` function returns the autorestart flag value of scope *scNum*. If the function detects an error, it returns `-1`.

Description

The `xPCScGetAutoRestart` function gets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (0) or enabled (1).

See Also

API functions `xPCScSetAutoRestart`

xPCScGetData

Purpose

Copy scope data to array

Prototype

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

Arguments

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

Description

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

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

See Also

API functions `xPCGetScope`, `xPCScGetState`, `xPCScGetSignals`

Scope object property `Data`

xPCScGetDecimation

Purpose

Return decimation of scope

Prototype

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

Arguments

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

Return

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

Description

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

See Also

API function `xPCScSetDecimation`

Scope object property `Decimation`

Purpose

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

Prototype

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

Arguments

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

Return

The `xPCScGetNumPrePostSamples` function returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this function returns the minimum integer value (`-2147483647-1`).

Description

The `xPCScGetNumPrePostSamples` function gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCScSetNumPrePostSamples`

Scope object property `NumPrePostSamples`

xPCScGetNumSamples

Purpose

Get number of samples in one data acquisition cycle

Prototype

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

Arguments

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

Return

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

Description

The `xPCScGetNumSamples` function gets the number of samples in one data acquisition cycle for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCScSetNumSamples`

Scope object property `NumSamples`

Purpose

Get number of signals in scope

Prototype

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

Arguments

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

Return

The `xPCScGetNumSignals` function returns the number of signals in the scope *scNum*. If the function detects an error, it returns `-1`.

Description

The `xPCScGetNumSignals` function gets the number of signals in the scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API function `xPCGetScope`

xPCScGetSignalList

Purpose

Copy list of signals to array

Prototype

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

Arguments

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

Description

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

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

Purpose

Copy list of signals to array

Prototype

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

Arguments

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

Description

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

Note This function will be 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

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

Return

The `xPCScGetStartTime` function returns the start time for the last data acquisition cycle of a scope. If the function detects an error, it returns `-1`.

Description

The `xPCScGetStartTime` function gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScGetNumSamples`, `xPCScGetDecimation`

Purpose

Get state of scope

Prototype

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

Arguments

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

Return

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

Description

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

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

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

See Also

API functions `xPCScStart`, `xPCScStop`

Scope object property `Status`

Purpose

Get trigger level for scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerLevel` function returns the scope trigger level. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerLevel` function gets the trigger level for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object property `TriggerLevel`

xPCScGetTriggerMode

Purpose

Get trigger mode for scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerMode` function returns the scope trigger mode. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerMode` function gets the trigger mode for scope *scNum*. Use the `xPCGetScope` function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope always triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only a user can trigger the scope. It is always possible for a user to trigger the scope; however, if you set the scope to this trigger mode, user intervention is the only way to trigger the scope. No other triggering is possible.

Constant	Value	Description
TRIGMD_SIGNAL	2	Signal must cross a value before the scope is triggered.
TRIGMD_SCOPE	3	Scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of triggerscopesample (see scopedata).

See Also

API functions xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode

Scope object method trigger

Scope object property TriggerMode

xPCScGetTriggerScope

Purpose

Get trigger scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerScope` function returns a trigger scope. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerScope` function gets the trigger scope for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object property `TriggerScope`

Purpose

Get sample number for triggering scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerScopeSample` function returns a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the function detects an error, it returns `INT_MIN` (`-2147483647-1`).

Description

The `xPCScGetTriggerScopeSample` function gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScSetTriggerScopeSample`

Scope object property `TriggerSample`

xPCScGetTriggerSignal

Purpose

Get trigger signal for scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerSignal` function returns the scope trigger signal. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerSignal` function gets the trigger signal for scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object method `trigger`

Scope object property `TriggerSignal`

Purpose

Get trigger slope for scope

Prototype

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

Arguments

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

Return

The `xPCScGetTriggerSlope` function returns the scope trigger slope. If the function detects an error, it returns `-1`.

Description

The `xPCScGetTriggerSlope` function gets the trigger slope of scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope. Use the constants defined in `xpcapiconst.h` to interpret the trigger slope. These constants have the following meanings:

Constant	Value	Description
`TRIGSLOPE_EITHER`	`0`	The trigger slope can be either rising or falling.
`TRIGSLOPE_RISING`	`1`	The trigger slope must be rising when the signal crosses the trigger value.
`TRIGSLOPE_FALLING`	`2`	The trigger slope must be falling when the signal crosses the trigger value.

See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object method `xpctarget.xpcsc.trigger`

Scope object properties `TriggerMode`, `TriggerSlope`

Purpose

Get type of scope

Prototype

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

Arguments

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

Return

The `xPCScGetType` function returns the scope type. If the function detects an error, it returns `-1`.

Description

The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the host computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCAddScope`, `xPCRemScope`

Scope object property `Type`

xPCScRemSignal

Purpose

Remove signal from scope

Prototype

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

Arguments

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

Description

The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCGetScopes` to determine the existing scopes, and use `xPCScGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

See Also

API functions `xPCScAddSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`, `xPCScGetSignals`, `xPCScGetState`

Scope object method `remsignal`

Purpose

Scope autorestart status

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
autorestart	Enter value to enable (`1`) or disable (`0`) scope autorestart.

Description

The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to `0` or `1`. `0` disables the flag, `1` enables it. Use this function only when the scope is stopped.

See Also

API functions `xPCScGetAutoRestart`

xPCScSetDecimation

Purpose

Set decimation of scope

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCScGetDecimation`, `xPCScGetState`

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

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

Description

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

See Also

API functions `xPCScGetNumPrePostSamples`, `xPCScGetState`

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

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

Description

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

See Also

API functions `xPCScGetNumSamples`, `xPCScGetState`

Scope object property `NumSamples`

Purpose

Set trigger level for scope

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Scope object property `TriggerLevel`

xPCScSetTriggerMode

Purpose

Set trigger mode of scope

Prototype

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

Arguments

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

Description

The `xPCScSetTriggerMode` function sets the trigger mode of scope *scNum* to *mode*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to interpret the trigger mode:

Constant	Value	Description
`TRIGMD_FREERUN`	`0`	The scope always triggers when it is ready to trigger, regardless of the circumstances. This is the default.
`TRIGMD_SOFTWARE`	`1`	Only a user can trigger the scope. It is always possible for a user to trigger the scope; however, if you set the scope to this trigger mode, user intervention is the only way to trigger the scope. No other triggering is possible.

Constant	Value	Description
TRIGMD_SIGNAL	2	Signal must cross a value before the scope is triggered.
TRIGMD_SCOPE	3	Scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of triggerscopesample (see scopedata).

See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScGetTriggerMode, xPCScGetState

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

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

Description

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

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

See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScGetState

Scope object property TriggerScope

Purpose

Set sample number for triggering scope

Prototype

```
void xPCScSetTriggerScopeSample(int port, int scNum, int
trigScSamp);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
trigScSamp	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.

Description

The `xPCScSetTriggerScopeSample` function sets the number of samples (*trigScSamp*) a triggering scope acquires before it triggers a second scope (*scNum*). Use the `xPCGetScopes` function to get a list of scopes.

For meaningful results, set *trigScSamp* between `-1` and (*nSamp*`-1`). *nSamp* is the number of samples in one data acquisition cycle for the triggering scope. However, no checking is done, and using a value that is too big causes the scope never to be triggered.

If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, enter a value of `-1` for *trigScSamp*.

See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetTriggerScopeSample`

Scope object properties `TriggerMode`, `TriggerSample`

xPCScSetTriggerSignal

Purpose

Select signal to trigger scope

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object property `TriggerSignal`

Purpose

Set slope of signal that triggers scope

Prototype

```
void xPCScSetTriggerSlope(int port, int scNum,
int trigSlope);
```

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
scNum	Enter the scope number.
trigSlope	Enter the slope mode for the signal that triggers the scope.

Description

The `xPCScSetTriggerSlope` function sets the trigger slope of scope *scNum* to *trigSlope*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to set the trigger slope:

Constant	Value	Description
`TRIGSLOPE_EITHER`	0	The trigger slope can be either rising or falling.
`TRIGSLOPE_RISING`	1	The trigger signal value must be rising when it crosses the trigger value.
`TRIGSLOPE_FALLING`	2	The trigger signal value must be falling when it crosses the trigger value.

See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Scope object property `TriggerSlope`

Purpose

Set software trigger of scope

Prototype

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

Arguments

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

Description

The `xPCScSoftwareTrigger` function triggers scope *scNum*. The scope must be in the state `Waiting for trigger` for this function to succeed. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

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

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

Description

The `xPCScStart` function starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire any 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

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

mode Valid values are

- `0` Turns the display off
- `1` Turns the display on

Description

On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the target application is stopped. When you turn the message display off, the message screen no longer updates.

See Also

API function `xPCGetEcho`

Purpose

Set last error to specific string constant

Prototype

```
void xPCSetLastError(int error);
```

Arguments

error	Specify the string constant for the error.

Description

The `xPCSetLastError` function sets the global error constant returned by `xPCGetLastError` to *error*. This is useful only to set the string constant to `ENOERR`, indicating no error was found.

See Also

API functions `xPCGetLastError`, `xPCErrorMsg`

xPCSetLoadTimeOut

Purpose

Change initialization timeout value between host computer and target computer

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
timeOut	Enter the new communication timeout value.

Description

The `xPCSetLoadTimeOut` function changes the timeout value for communication between the host computer and target computer. The *timeOut* value is the time an 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

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
logging_data	Logging mode and increment value.

Description

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

See Also

API function `xPCGetLogMode`

API structure `lgmode`

Target object property `LogMode`

xPCSetParam

Purpose

Change value of parameter

Prototype

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

Arguments

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

Description

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

See Also

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

Purpose

Change target application sample time

Prototype

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

Arguments

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
ts	Sample time for the target application.

Description

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

See Also

API function `xPCGetSampleTime`

Target object property `SampleTime`

xPCSetScope

Purpose

Set properties of scope

Prototype

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

Arguments

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

Description

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

port	Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
tfinal	Enter the stop time, in seconds.

Description

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

See Also

API function `xPCGetStopTime`

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 all parameter changes 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

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

Arguments

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

Return

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

Description

The `xPCTargetPing` function pings the target computer and returns `1` or `0` depending on whether the target responds or not. This function returns an error string constant only when there is an error in the input parameter (for example, the port number is invalid or *`port`* is not open). Other errors, such as the inability to connect to the target, are ignored.

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

See Also

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

Purpose

Get status of grid line for particular scope

Prototype

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

Arguments

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

Return

Returns the status of the grid for a scope of type `SCTYPE_TARGET`. If the function detects an error, it returns `-1`.

Description

The `xPCTgScGetGrid` function gets the state of the grid lines for scope *scNum* (which must be of type `SCTYPE_TARGET`). A return value of `1` implies grid on, while `0` implies grid off. Note that when the scope mode is set to `SCMODE_NUMERICAL`, the grid is not drawn even when the `grid` mode is set to `1`.

Tip

- Use `xPCTgScSetMode` and `xPCTgScGetMode` to set and retrieve the scope mode.
- Use `xPCGetScopes` to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

xPCTgScGetMode

Purpose

Get scope mode for displaying signals

Prototype

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

Arguments

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

Return

The `xPCTgScGetMode` function returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

If this function detects an error, it returns `-1`.

Description

The `xPCTgScGetMode` function gets the mode (`SCMODE_NUMERICAL`, `SCMODE_REDRAW`, `SCMODE_SLIDING`, `SCMODE_ROLLING`) of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`

Purpose

Get view mode for target computer display

Prototype

```
int xPCTgScGetViewMode(int port);
```

Arguments

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

Return

The `xPCTgScGetViewMode` function returns the view mode for the target computer screen. If the function detects an error, it returns `-1`.

Description

The `xPCTgScGetViewMode` function gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is of the scope currently displayed on the screen. If the value is `0`, then all defined scopes are currently displayed on the target computer screen. In the latter case, no scopes are in focus (that is, all scopes are unzoomed).

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Target object property `ViewMode`

xPCTgScGetYLimits

Purpose

Copy *y*-axis limits for scope to array

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`

Scope object property `YLimit`

Purpose

Set grid mode for scope

Prototype

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

Arguments

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

Description

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

See Also

API functions `xPCGetScopes`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Grid`

xPCTgScSetMode

Purpose

Set display mode for scope

Prototype

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

Arguments

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

Description

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

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

Use the `xPCGetScopes` function to get a list of scopes.

See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`

Purpose

Set view mode for scope

Prototype

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

Arguments

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

Description

The `xPCTgScSetViewMode` function sets the target computer screen to display one scope with scope number *scNum*. If you set *scNum* to `0`, the target computer screen displays all the 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`

4

xPC Target API Reference for COM

COM API Methods

In this section...

Note The xPC Target COM API is no longer being maintained. You should use the xPC Target API for Microsoft .NET API Framework instead. See "xPC Target API for Microsoft .NET Framework"

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

Name	This value contains the name of the file or folder.
Date	This value contains the date the file or folder was last modified.
Time	This value contains the time the file or folder was last modified.
Bytes	This value contains the size of the file in bytes. If the element is a folder, this value is 0.
isdir	This value indicates if the element is a file (0) or folder (1). If it is a folder, *Bytes* has a value of 0.

Description

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

See Also

API methodxPCFileSystem.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

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

FATType	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
FATCount	This value contains the number of FAT partitions on the volume.
MaxDirEntries	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
BytesPerSector	This value contains the sector size. This value is most likely to be 512.
SectorsPerCluster	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
TotalClusters	This value contains the number of file storage clusters on the volume.
BadClusters	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
FreeClusters	This value contains the number of clusters that are currently available for storage.
Files	This value contains the number of files, including directories, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
FileChains	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of *Files*.

FreeChains	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
LargestFreeChain	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to *FreeClusters*.

Description

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

See Also

API method `xPCFileSystem.GetDiskInfo`

xPCFileSystem.CD

Purpose

Change current folder on target computer to specified path

Prototype

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

See Also

API method `xPCFileSystem.PWD`

Purpose

Close file on target computer

Prototype

```
CloseFile(long filehandle);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

See Also

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

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

`GetDiskInfo(BSTR driveLetter);`

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Description

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

See Also

API structure `FSDiskInfo`

API method `xPCFileSystem.DirList`

xPCFileSystem.GetFileSize

Purpose

Return size of file on target computer

Prototype

```
long GetFileSize(long filehandle);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

The `xPCFileSystem.GetFileSize` method returns the size, in bytes, of the file associated with *`filehandle`* on the target computer. *`filehandle`* is the handle of a file previously opened by the `xPCFileSystem.OpenFile` method.

See Also

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

Purpose

Initialize file system object to communicate with target computer

Prototype

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

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

Return

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

Description

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

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

See Also

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

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 fileHandle, int start, int numbytes);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

The `xPCFileSystem.ReadFile` method reads an open file on the target computer and returns the results of the read operation as a `VARIANT` of type `Byte`. *`fileHandle`* is the file handle of a file previously opened by `xPCFileSystem.OpenFile`. You can specify that the read operation begin at the beginning of the file (default) or at a certain offset into the file (*`start`*). The *`numbytes`* parameter specifies how many bytes the `xPCFileSystem.ReadFile` method is to read from the file.

See Also

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

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

```
long RMDIR(BSTR dirname);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

xPCFileSystem.ScGetFileName

Purpose

Get name of file for scope

Prototype

```
BSTR ScGetFileName(long scNum);
```

Member Of

```
XPCAPICOMLib.xPCFileSystem
```

Arguments

`[in] scNum`	Enter the scope number.

Return

Returns the name of the file for the scope.

Description

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

See Also

API method `xPCFileSystem.ScSetFileName`

Purpose

Get write mode of file for scope

Prototype

```
long ScGetWriteMode(long scNum);
```

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

This method returns the number indicating the write mode. Values are

0	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
1	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always has 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

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

The `xPCFileSystem.ScSetFileName` method sets the name of the file to which the scope will save the signal data. The 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 `xPCFileSystem.ScGetFileName`

xPCFileSystem.ScSetWriteMode

Purpose

Specify when file allocation table entry is updated

Prototype

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

`0`	Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
`1`	Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system always has 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

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

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

See Also

API method `xPCFileSystem.ScGetWriteSize`

Scope object property `WriteSize`

xPCFileSystem.WriteFile

Purpose

Write to file on target computer

Prototype

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

Member Of

`XPCAPICOMLib.xPCFileSystem`

Arguments

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

Return

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

Description

The `xPCFileSystem.WriteFile` method writes the contents of the `VARIANT` `buffer`, of type `Byte`, to the file specified by `fileHandle` on the target computer. The `fileHandle` parameter is the handle of a file previously opened by `xPCFSOpenFile`. `numbytes` is the number of bytes to write to the file.

See Also

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

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

```
long GetLoadTimeOut();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

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

Description

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

When you load a new target application onto the target computer, the method `xPCTarget.LoadApp` waits for a certain amount of time before checking to see whether the initialization of the target application is complete. In the case where initialization of the target application is not complete, the method `xPCTarget.LoadApp` returns a timeout error. By default, `xPCTarget.LoadApp` checks five times to see whether the target application is ready, with each attempt taking about 1 second. However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout is generated. The method `xPCProtocol.SetLoadTimeOutxPCProtocol.SetLoadTimeOut` sets the timeout to a different number.

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

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

```
long Init();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

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

Description

The `xPCProtocol.Init` method initializes the xPC Target API by loading the xPC Target DLL, `xpcapi.dll`, into memory. To load `xpcapi.dll` into memory, the method requires that the `xpcapi.dll` file be in one of the following directories:

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

Purpose

Return error status

Prototype

```
long isxPCError();
```

Member Of

`XPCAPICOMLIB.xPCProtocol`

Return

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

Description

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

See Also

API function `xPCProtocol.GetxPCErrorMsg`

xPCProtocol.Port

Purpose

Contain communication channel index

Prototype

```
long Port();
```

Member Of

`XPCAPICOMLIB.xPCProtocol`

Return

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

Description

The `xPCProtocol.Port` property contains the communication channel index if connection with the target computer succeeds. Note that you only need to use this property when working with a model-specific COM library that you generate from a Simulink model. See "Model-Specific COM Interface Library (model_nameCOMiface.dll)".

Purpose

Reboot target computer

Prototype

```
long Reboot();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

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

Description

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

xPCProtocol.RS232Connect

Purpose

Open RS-232 connection to target computer

Prototype

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

Member Of

`XPCAPICOMLib.xPCProtocol`

Arguments

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

Return

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

Description

The `xPCProtocol.RS232Connect` method initiates an RS-232 connection to an 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.

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

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

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

Purpose

Open TCP/IP connection to target computer

Prototype

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

Member Of

`XPCAPICOMLIB.xPCProtocol`

Arguments

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

Return

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

Description

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

xPCProtocol.Term

Purpose

Unload xPC Target API DLL from memory

Prototype

```
long Term();
```

Member Of

`XPCAPICOMLib.xPCProtocol`

Return

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

Description

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

Purpose

Create new file scope

Prototype

```
long AddFileScope(long scNum);
```

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

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

Return

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

Description

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

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

xPCScopes.AddHostScope

Purpose

Create new host scope

Prototype

```
long AddHostScope(long scNum);
```

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

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

Return

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

Description

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

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

Purpose

Create new target scope

Prototype

```
long AddTargetScope(long scNum);
```

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

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

Return

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

Description

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

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

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 reported no 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

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

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

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

Return

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

Description

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

Purpose

Get data acquisition status for scope

Prototype

```
long IsScopeFinished(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum* Enter the scope number.

Return

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

Description

The `xPCScopeos.IsScopeFinished` method gets a `1` or `0` depending on whether scope *scNum* is finished (state of `SCST_FINISHED`) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state.

xPCScopes.isxPCError

Purpose

Get error status

Prototype

```
long isxPCError();
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Return

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

Description

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

See Also

API function `xPCScopes.GetxPCError`

Purpose

Remove scope

Prototype

```
long RemScope(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

xPCScopes.ScopeAddSignal

Purpose

Add signal to scope

Prototype

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

Member Of

`XPCAPICOMLib.xPCScopes`

Arguments

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

Return

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

Description

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

Purpose

Scope autorestart value

Prototype

```
long ScopeGetAutoRestart(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

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

Return

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

Description

The `xPCScopes.ScopeGetData` method gets the data used in a scope. Use this function for scopes of type `SCTYPE_HOST`. The scope must be either in state `Finished` or in state `Interrupted` for the data to be retrievable. (Use the `xPCScopes.ScopeGetState` method to check the state of the scope.) The data must be retrieved one signal at a time. The calling function determines and allocates the space ahead of time to store the scope data. Use the method `xPCScopes.ScopeGetSignals` to get the list of signals in the scope for *signal_id*.

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

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 `N`th sample is acquired in a scope window.

Purpose

Get number of pre- or posttriggering samples before triggering scope

Prototype

```
long ScopeGetNumPrePostSamples(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

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

Return

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

Description

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

xPCScopes.ScopeGetStartTime

Purpose

Get last data acquisition cycle start time

Prototype

```
double ScopeGetStartTime(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

Purpose

Get state of scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum* Enter the scope number.

Return

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

Description

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

Constants to find the scope state have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

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 always triggers when it is ready to trigger, regardless of the circumstances.
`TRIGMD_SOFTWARE`	`1`	Only a user can trigger the scope. It is always possible for a user to trigger the scope; however, if you set the scope to this trigger mode, user intervention is the only way to trigger the scope. No other triggering is possible.

Constant	Value	Description
TRIGMD_SIGNAL	2	Signal must cross a value before the scope is triggered.
TRIGMD_SCOPE	3	Scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of triggerscopesample (see scopedata).

See Also

API function xPCScopes.ScopeGetTriggerModeStr

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 always triggers when it is ready to trigger, regardless of the circumstances.
`Software`	Only a user can trigger the scope. It is always possible for a user to trigger the scope; however, if you set the scope to this trigger mode, user intervention is the only way to trigger the scope. No other triggering is possible.
`Signal`	Signal must cross a value before the scope is triggered.
`Scope`	Scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of `triggerscopesample` (see `scopedata`).

See Also

API function `xPCScopes.ScopeGetTriggerMode`

Purpose

Get sample number for triggering scope

Prototype

```
long ScopeGetTriggerSample(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.ScopeGetTriggerSample` method returns a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the method detects an error, it returns `-1`.

Description

The `xPCScopes.ScopeGetTriggerSample` method gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and `-1` for the special case where triggering is at the end of the data acquisition cycle for a triggering scope.

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

Return

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

Description

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

Purpose

Scope autorestart value

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

xPCScopes.ScopeSetDecimation

Purpose

Set decimation of scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

The `xPCScopes.ScopeSetDecimation` method sets the *decimation* of scope *scNum*. The decimation is a number, `N`, meaning every `N`th sample is acquired in a scope window. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

Purpose

Set number of pre- or posttriggering samples before triggering scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

xPCScopes.ScopeSetNumSamples

Purpose

Set number of samples in one data acquisition cycle

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

Purpose

Set trigger level for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

xPCScopes.ScopeSetTriggerMode

Purpose

Set trigger mode of scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

Use the constants defined here to interpret the trigger mode:

Constant	Value	Description
`TRIGMD_FREERUN`	`0`	The scope always triggers when it is ready to trigger, regardless of the circumstances. This is the default.
`TRIGMD_SOFTWARE`	`1`	Only a user can trigger the scope. It is always possible for a user to trigger the scope; however, if you set the scope to this trigger mode, user intervention is the only way to trigger the scope. No other triggering is possible.

Constant	Value	Description
TRIGMD_SIGNAL	2	Signal must cross a value before the scope is triggered.
TRIGMD_SCOPE	3	Scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of triggerscopesample (see scopedata).

xPCScopes.ScopeSetTriggerSample

Purpose

Set sample number for triggering scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

For meaningful results, set *`trigScSample`* between `-1` and (*`nSamp`*`-1`). *`nSamp`* is the number of samples in one data acquisition cycle for the triggering scope. However, no checking is done, and using a value that is too big causes the scope never to be triggered.

If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, use a value of `-1` for *`trigScSamp`*.

Purpose

Select signal to trigger scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

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

Use the constants defined here to set the trigger slope:

Constant	Value	Description
`TRIGSLOPE_EITHER`	`0`	The trigger slope can be either rising or falling.
`TRIGSLOPE_RISING`	`1`	The trigger signal value must be rising when it crosses the trigger value.
`TRIGSLOPE_FALLING`	`2`	The trigger signal value must be falling when it crosses the trigger value.

Purpose

Set software trigger of scope

Prototype

```
long ScopeSoftwareTrigger(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

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

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 any 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

```
long TargetScopeGetMode(long scNum);
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

The `xPCScopes.TargetScopeGetMode` method returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

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

Description

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

See Also

API function `xPCScopes.TargetScopeGetModeStr`

xPCScopes.TargetScopeGetModeStr

Purpose

Get scope mode string for displaying signals

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

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

Description

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

See Also

API function `xPCScopes.TargetScopeGetMode`

Purpose

Get view mode for target computer display

Prototype

```
long TargetScopeGetViewMode();
```

Member Of

`XPCAPICOMLIB.xPCScopes`

Return

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

Description

The `xPCScopes.TargetScopeGetViewMode` method gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is of the scope currently displayed on the screen. If the value is `0`, then all defined scopes are currently displayed on the target computer screen. In the latter case, no scopes are in focus (that is, all scopes are unzoomed).

xPCScopes.TargetScopeGetYLimits

Purpose

Get *y*-axis limits for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

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

Purpose

Set grid mode for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

xPCScopes.TargetScopeSetMode

Purpose

Set display mode for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

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

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Set view mode for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

[in] *scNum*	Enter the scope number.

Return

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

Description

The `xPCScopes.TargetScopeSetViewMode` method sets the target computer screen to display one scope with scope number *scNum*. If you set *scNum* to `0`, the target computer screen displays all the scopes. Use the `xPCScopes.GetScopes` method to get a list of scopes.

xPCScopes.TargetScopeSetYLimits

Purpose

Set *y*-axis limits for scope

Prototype

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

Member Of

`XPCAPICOMLIB.xPCScopes`

Arguments

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

Return

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

Description

The `xPCScopes.TargetScopeSetYLimits` method sets the *y*-axis limits for a scope with scope number *scNum* and type `SCTYPE_TARGET` to the values in the double array *YlimitArray*. The first element is the lower limit, and the second element is the upper limit. Set both limits to `0.0` to specify autoscaling. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Purpose

Get average task execution time

Prototype

```
double AverageTET();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

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, *`model_name`*, in a `printf` or similar statement. In case of error, the string is unchanged. Be sure to allocate enough space to accommodate the longest target name you have.

Purpose

Get execution time for target application

Prototype

```
double GetExecTime();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

xPCTarget.GetNumOutputs

Purpose

Get number of outputs

Prototype

```
long GetNumOutputs();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

The `xPCTarget.GetNumOutputs` method gets the number of outputs in the target application. The number of outputs equals the sum of the input signal widths of all output blocks at the root level of the Simulink model.

Purpose

Get number of tunable parameters

Prototype

```
long GetNumParams();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

xPCTarget.GetNumSignals

Purpose

Get number of signals

Prototype

```
long GetNumSignals();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

Purpose

Get number of states

Prototype

```
long GetNumStates();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

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

Return

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

Description

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

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

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

Purpose

Get parameter values

Prototype

```
VARIANT GetParam(long paramIdx);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

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

Return

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

Description

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

See Also

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

Microsoft Visual Basic .NET 2003 solution located in `C:\matlabroot\toolbox\rtw\targets\xpc\api\VBNET\SigsAndParamsDemo`

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

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

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

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

Return

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

Description

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

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

```
double GetSampleTime();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

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.

Purpose

Get signal IDs from signal label

Prototype

```
VARIANT GetSignalidsfromLabel(BSTR sigLabel);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

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

Return

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

Description

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

This method assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the "Signal Properties Controls"). Note that the 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.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] *NumOfSignals*	Enter the number of signals to acquire (the number of IDs in *SignalsIdxArray*).
[out] *SignalsIdxArray*	Enter the IDs of the signals to acquire.

Return

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

Description

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

Tip

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

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

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

Return

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

Description

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

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

Purpose

Get stop time

Prototype

```
double GetStopTime();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

xPCTarget.GetTETLog

Purpose

Get TET log

Prototype

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

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

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

Return

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

Description

The `xPCTarget.GetTETLog` method gets the task execution time (TET) log. Entering `1` for *decimation* copies all values. Entering `N` copies every 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 start, long numsamples,
long decimation);
```

Member Of

`XPCAPICOMLib.xPCTarget`

Arguments

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

Return

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

Description

The `xPCTarget.GetTimeLog` method gets the time log. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering `1` for *`decimation`* copies all values. Entering `N` copies every `N`th value. For *`start`*, the sample indices range from `0` to (`N`-1), where `N` is the return value of `xPCTarget.NumLogSamples`. Use the `xPCTarget.NumLogSamples` method to get the number of samples.

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

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 reported no 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

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

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

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

Description

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

xPCTarget.IsAppRunning

Purpose

Return running status for target application

Prototype

```
long IsAppRunning();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

Purpose

Return overload status for target computer

Prototype

```
long IsOverloaded();
```

Member Of

`XPCAPICOMLib.xPCTarget`

Return

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

Description

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

xPCTarget.isxPCError

Purpose

Return error status

Prototype

```
long isxPCError();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

See Also

API method `xPCTarget.GetxPCError`

Purpose

Load target application onto target computer

Prototype

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

Member Of

`XPCAPICOMLIB.xPCTarget`

Arguments

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

Return

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

Description

The `xPCTarget.LoadApp` method loads the compiled target application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to NULL or to the string `'nopath'` if the application is in the current folder. The variable *filename* must not contain the target application extension.

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

be generated. The methods `xPCProtocol.GetLoadTimeOut` and `xPCProtocol.SetLoadTimeOut` control the number of attempts made.

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

```
long MaxLogSamples();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

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

Purpose

Copy minimum task execution time to array

Prototype

```
VARIANT MinimumTET();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

xPCTarget.NumLogSamples

Purpose

Return number of samples in log buffer

Prototype

```
long NumLogSamples();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

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

Purpose

Return number of times log buffer wraps

Prototype

```
long NumLogWraps();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

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

xPCTarget.SetParam

Purpose

Change parameter value

Prototype

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

Member Of

`XPCAPICOMLIB.xPCTarget`

Arguments

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

Return

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

Description

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

See Also

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

Purpose

Change sample time for target application

Prototype

```
long SetSampleTime(double ts);
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Arguments

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

Return

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

Description

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

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

```
long StopApp();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

Purpose

Unload target application

Prototype

```
long UnLoadApp();
```

Member Of

`XPCAPICOMLIB.xPCTarget`

Return

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

Description

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

5

Configuration Parameters

This topic deals with configuration parameters in xPC Target Explorer and in the MATLAB API.

Setting Configuration Parameters

In this section...

xPC Target options Pane

Set up general information about building target applications, including target, execution, data logging, and other options.

Configuration

To enable the xPC Target options pane, you must:

1. Select `xpctarget.tlc` or `xpctargetert.tlc` for the **System target file** parameter on the code generation pane.
2. Select `C` for the **Language** parameter on the code generation pane.

Tips

- The default xPC Target options work for the generation of most target applications. If you want to customize the build of your target application, set the option parameters to suit your specifications.
- To access these parameters from the MATLAB command line, use:
  - `gcs` — To access the current model.
  - `set_param` — To set the parameter value.
  - `get_param` — To get the current value of the parameter.

See Also

"xPC Target Options Configuration Parameter"

Automatically download application after building

Enable Simulink Coder to build and download the target application to the target computer.

Settings

Default: on

 On

Builds and downloads the target application to the target computer.

 Off

Builds the target application, but does not download it to the target computer.

Command-Line Information

Parameter: `xPCisDownloadable`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

"Build and Download Target Application"

Download to default target PC

Direct Simulink Coder to download the target application to the default target computer.

Settings

Default: on

 On

Downloads the target application to the default target computer. Assumes that you configured a default target computer through xPC Target Explorer.

 Off

Enables the **Specify target PC name** field so that you can enter the target computer to which to download the target application.

Dependency

This parameter enables **Specify target PC name**.

Command-Line Information

Parameter: `xPCisDefaultEnv`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

- "Network Communication Setup"
- "Serial Communication Setup"

Specify target PC name

Specify a target computer name for your target application.

Settings

`''`

Tip

The target computer name appears in xPC Target Explorer as the target computer node, for example `TargetPC1`.

Dependencies

This parameter is enabled by **Download to default target PC**.

Command-Line Information

Parameter: `xPCTargetPCEnvName`
Type: string
Value: Any valid target computer
Default: `''`

See Also

"xPC Target Explorer"

Name of xPC Target object created by build process

Enter the name of the target object created by the build process.

Settings

Default: `tg`

Tip

Use this name when you work with the target object through the command-line interface.

Command-Line Information

Parameter: `RL32ObjectName`
Type: string
Value: `'tg'` | valid target object name
Default: `'tg'`

See Also

"Target Driver Objects"

Use default communication timeout

Direct xPC Target software to wait 5 (default) seconds for the target application to be downloaded to the target computer.

Settings

Default: on

 On

Waits the default amount of seconds (5) for the target application to be downloaded to the target computer.

 Off

Enables the **Specify the communication timeout in seconds** field so that you can enter the maximum length of time in seconds you want to wait for a target application to be downloaded to the target computer.

Dependencies

This parameter enables **Specify the communication timeout in seconds**.

Command-Line Information

Parameter: `xPCisModelTimeout`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

"Increase the Time for Downloads"

Specify the communication timeout in seconds

Specify a timeout, in seconds, to wait for the target application to download to the target computer.

Settings

Default: `5`

Tip

Enter the maximum length of time in seconds you want to allow the xPC Target software to wait for the target application to download to the target computer. If the target application is not downloaded within this time frame, the software generates an error.

Dependencies

This parameter is enabled by **Use default communication timeout**.

Command-Line Information

Parameter: `xPCModelTimeoutSecs`
Type: string
Value: Any valid number of seconds
Default: `'5'`

See Also

"Increase the Time for Downloads"

Execution mode

Specify target application execution mode.

Settings

Default: `Real-Time`

`Real-Time`
: Executes target application in real time.

`Freerun`
: Runs the target application as fast as possible.

Command-Line Information

Parameter: `RL32ModeModifier`
Type: string
Value: `'Real-Time' | 'Freerun'`
Default: `'Real-Time'`

See Also

"Set Configuration Parameters"

Real-time interrupt source

Select a real-time interrupt source from the I/O board.

Settings

Default: `Timer`

`Timer`
: Specifies that the board interrupt source is a timer.

`Auto (PCI only)`
: Enables the xPC Target software to automatically determine the IRQ that the BIOS assigned to the board and use it.

`3` to `15`
: Specifies that the board interrupt source is an IRQ number on the board.

Tips

- The `Auto (PCI only)` option is available only for PCI boards. If you have an ISA board (PC 104 or onboard parallel port), you must set the IRQ manually.
- The xPC Target software treats PCI parallel port plug-in boards like ISA boards. For PCI parallel port plug-in boards, you must set the IRQ manually.
- Multiple boards can share the same interrupt number.

Command-Line Information

Parameter: `RL32IRQSourceModifier`
Type: string
Value: `'Timer' | Auto (PCI only) | '3'|'4'|'5' | '6'|'7' |'8' |'9' |'10' |'11' |'12' |'13' |'14' |'15'`
Default: `'Timer'`

See Also

"Set Configuration Parameters"

I/O board generating the interrupt

Specify the board interrupt source.

Settings

Default: `None/Other`

`ATI-RP-R5`
: Specifies that the interrupt source is an ATI-RP-R5 board.

`AudioPMC+`
: Specifies that the interrupt source is the Bittware AudioPMC+ audio board.

`Bitflow NEON`
: Specifies that the interrupt source is the Bitflow NEON video board.

`CB_CIO-CTR05`
: Specifies that the interrupt source is the Measurement Computing CIO-CTR05 board.

`CB_PCI-CTR05`
: Specifies that the interrupt source is the Measurement Computing PCI-CTR05 board.

`Diamond_MM-32`
: Specifies that the interrupt source is the Diamond Systems MM-32 board.

`FastComm 422/2-PCI`
: Specifies that the interrupt source is the FastComm 422/2-PCI board.

`FastComm 422/2-PCI-335`
: Specifies that the interrupt source is the FastComm 422/2-PCI-335 board.

`FastComm 422/4-PCI-335`
: Specifies that the interrupt source is the FastComm 422/4-PCI-335 board.

`GE_Fanuc(VMIC)_PCI-5565`
: Specifies that the interrupt source is the GE Fanuc VMIC PCI-5565 board.

`General Standards 24DSI12`
: Specifies that the interrupt source is the General Standards 24DSI12 board.

`Parallel_Port`
: Specifies that the interrupt source is the parallel port of the target computer.

`Quatech DSCP-200/300`
: Specifies that the interrupt source is the Quatech DSCP-200/300 board.

`Quatech ESC-100`
: Specifies that the interrupt source is the Quatech ESC-100 board.

`Quatech QSC-100`
: Specifies that the interrupt source is the Quatech QSC-100 board.

`Quatech QSC-200/300`
: Specifies that the interrupt source is the Quatech QSC-200/300 board.

`RTD_DM6804`
: Specifies that the interrupt source is the Real-Time Devices DM6804 board.

`SBS_25x0_ID_0x100`
: Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x100.

`SBS_25x0_ID_0x101`
: Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x101.

`SBS_25x0_ID_0x102`
: Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x102.

`SBS_25x0_ID_0x103`
: Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x103.

`Scramnet_SC150+`
: Specifies that the interrupt source is the Systran Scramnet+ SC150 board.

`Softing_CAN-AC2-104`
: Specifies that the interrupt source is the Softing CAN-AC2-104 board.

`Softing_CAN-AC2-PCI`
Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

`Speedgoat_IO301`
Specifies that the interrupt source is the Speedgoat IO301 FPGA board.

`Speedgoat_IO302`
Specifies that the interrupt source is the Speedgoat IO302 FPGA board.

`Speedgoat_IO303`
Specifies that the interrupt source is the Speedgoat IO303 FPGA board.

`Speedgoat_IO311`
Specifies that the interrupt source is the Speedgoat IO311 FPGA board.

`Speedgoat_IO312`
Specifies that the interrupt source is the Speedgoat IO312 FPGA board.

`Speedgoat_IO313`
Specifies that the interrupt source is the Speedgoat IO313 FPGA board.

`Speedgoat_IO314`
Specifies that the interrupt source is the Speedgoat IO314 FPGA board.

`Speedgoat_IO325`
Specifies that the interrupt source is the Speedgoat IO325 FPGA board.

`UEI_MFx`
Specifies that the interrupt source is a United Electronic Industries UEI-MF series board.

`None/Other`
Specifies that the I/O board has no interrupt source.

Command-Line Information

Parameter: `xPCIRQSourceBoard`
Type: string
Value: `'ATI-RP-R5' |`
`'AudioPMC+' |`
`'Bitflow NEON' |`
`'CB_CIO-CTR05' |`
`'CB_PCI-CTR05' |`
`'Diamond_MM-32' |`
`'FastComm 422/2-PCI' |`

```
'FastComm 422/2-PCI-335' |
'FastComm 422/4-PCI-335' |
'GE_Fanuc(VMIC)_PCI-5565' |
'General Standards 24DSI12' |
'Parallel_Port' |
'Quatech DSCP-200/300' |
'Quatech ESC-100' |
'Quatech QSC-100' |
'Quatech QSC-200/300' |
'RTD_DM6804' |
'SBS_25x0_ID_0x100' |
'SBS_25x0_ID_0x101' |
'SBS_25x0_ID_0x102' |
'SBS_25x0_ID_0x103' |
'Scramnet_SC150+' |
'Softing_CAN-AC2-104' |
'Softing_CAN-AC2-PCI' |
'Speedgoat_IO301' |
'Speedgoat_IO302' |
'Speedgoat_IO303' |
'Speedgoat_IO311' |
'Speedgoat_IO312' |
'Speedgoat_IO313' |
'Speedgoat_IO314' |
'Speedgoat_IO325' |
'UEI_MFx' |
'None/Other'
```

Default: `'None/Other'`

See Also

"Set Configuration Parameters"

PCI slot (-1: autosearch) or ISA base address

Enter the slot number or base address for the I/O board generating the interrupt.

Settings

Default: -1

The PCI slot can be either `-1` (let the xPC Target software determine the slot number) or of the form `[bus, slot]`.

The base address is a hexadecimal number of the form `0x300`.

Tip

To determine the bus and PCI slot number of the boards in the target computer, type `getxpcpci` in the MATLAB window.

Command-Line Information

Parameter: `xPCIOIRQSlot`
Type: string
Value: `'-1'` | hexadecimal value
Default: `'-1'`

See Also

"xPC Target Options Configuration Parameter"

"PCI Bus I/O Devices"

Log Task Execution Time

Log task execution times to the target object property `tg.TETlog`.

Settings

Default: on

 On

Logs task execution times to the target object property `tg.TETlog`.

 Off

Does not log task execution times to the target object property `tg.TETlog`.

Command-Line Information

Parameter: `RL32LogTETModifier`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

"xPC Target Options Configuration Parameter"

"Signal Logging Basics"

Signal logging data buffer size in doubles

Enter the maximum number of sample points to save before wrapping.

Settings

Default: 100000

The maximum value for this option can be no more than the available target computer memory, which the xPC Target software also uses to hold other items.

Tips

- Target applications use this buffer to store the time, states, outputs, and task execution time logs as defined in the Simulink model.
- The maximum value for this option derives from available target computer memory, which the xPC Target software also uses to hold other items. For example, in addition to signal logging data, the software also uses the target computer memory for the xPC Target kernel, target application, and scopes.

  For example, assume the model `my_xpc_osc2` has six data items (one time, two states, two outputs, and one task execution time (TET)). If you enter a buffer size of `100000`, the target object property `tg.MaxLogSamples` is calculated as `floor(100000) / 6) = 16666`. After saving 16666 sample points, the buffer wraps and further samples overwrite the older ones.
- If you enter a logging buffer size larger than the available RAM on the target computer, after downloading and initializing the target application, the target computer displays a message, `ERROR: allocation of logging memory failed`. In this case you need to install more RAM or reduce the buffer size for logging. In any case you must reboot the target computer. To calculate the maximum buffer size you might have for you target application logs, divide the amount of available RAM on your target computer by 8. Enter that value for the **Signal logging data buffer size in doubles** value.

Command-Line Information

Parameter: `RL32LogBufSizeModifier`
Type: string

Value: `'100000'` | any valid memory size
Default: `'100000'`

See Also

"xPC Target Options Configuration Parameter"

Enable profiling

Enable profiling and visual presentation of target computer execution.

Settings

Default: off

 On

Profile target computer execution.

 Off

Do not profile target computer execution.

Tips

- Before building and downloading a model, select this check box to observe the target computer thread execution.
- If you are using multiple CPU cores on a target computer, select **Enabling profiling** to verify that the xPC Target software is actually executing on the multiple CPU cores.

Tip For more on configuring your model for concurrent execution, see "Design Considerations".

Command-Line Information

Parameter: `xPCTaskExecutionProfile`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Profiling Target Application Execution"

Number of events (each uses 20 bytes)

Enter the maximum of events to log for the profiling tool.

Settings

Default: 5000

The maximum number of events to be logged for the profiling tool.

Tips

- An event is the start of end of an interrupt or iteration of the model. For example, one sample can four events: the beginning and end of an interrupt, and the beginning and end of an iteration.
- Use this parameter in conjunction with the **Enable profiling** parameter.
- Each event contains information such as the CPU ID, model thread ID (TID), event ID, and time stamp readings. Each event occupies 20 bytes.

Command-Line Information

Parameter: `xPCRL32EventNumber`
Type: string
Value: any valid number of events
Default: `'5000'`

See Also

"Profiling Target Application Execution"

Double buffer parameter changes

Use a double buffer for parameter tuning. This enables parameter tuning so that the process of changing parameters in the target application uses a double buffer.

Settings

Default: off

 On

Changes parameter tuning to use a double buffer.

 Off

Suppresses double buffering of parameter changes in the target application.

Tips

- When a parameter change request is received, the new value is compared to the old one. If the new value is identical to the old one, it is discarded, and if different, queued.
- At the start of execution of the next sample of the real-time task, all queued parameters are updated. This means that parameter tuning affects the task execution time (TET), and the very act of parameter tuning can cause a CPU overload error.
- Double buffering leads to a more robust parameter tuning interface, but it increases Task Execution Time (TET) and the higher probability of overloads. Under typical conditions, keep double buffering off (default).

Command-Line Information

Parameter: `xpcDblBuff`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"xPC Target Options Configuration Parameter"

Load a parameter set from a file on the designated target file system

Automatically load a parameter set from a file on the designated target computer file system.

Settings

Default: off

 On

Enable the automatic loading of a parameter set from the file specified by **File name** on the designated target computer file system.

 Off

Suppress the automatic loading of a parameter set from a file on the designated target computer file system.

Dependencies

This parameter enables **File name**.

Command-Line Information

Parameter: `xPCLoadParamSetFile`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"xPC Target Options Configuration Parameter"

"Save and Reload Parameters with MATLAB Language"

File name

Specify the target computer file name from which to load the parameter set.

Settings

`''`

Tip

If the named file does not exist, the software loads the parameter set built with the model.

Dependencies

This parameter is enabled by **Load a parameter set from a file on the designated target file system**.

Command-Line Information

Parameter: `xPCOnTgtParamSetFileName`
Type: string
Value: Any valid file name
Default: `''`

See Also

"xPC Target Options Configuration Parameter"

Build COM objects from tagged signals/parameters

Enable build process to create a model-specific COM library file.

Settings

Default: off

 On

Creates a model-specific COM library file, `<model_name>COMiface.dll`.

 Off

Does not create a model-specific COM library file.

Tip

Use the model-specific COM library file to create custom GUIs with Visual Basic or other tools that can use COM objects.

Command-Line Information

Parameter: `xpcObjCom`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Creating the Target Application and Model-Specific COM Library"

Generate CANape extensions

Enable target applications to generate data, such as that for A2L, for Vector CANape.

Settings

Default: off

 On

Enables target applications to generate data, such as that for A2L, for Vector CANape.

 Off

Does not enable target applications to generate data, such as that for A2L, for Vector CANape.

Command-Line Information

Parameter: `xPCGenerateASAP2`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Configuring the Vector CANape® Device"

Include model hierarchy on the target application

Includes the Simulink model hierarchy as part of the target application.

Settings

Default: off

 On

Includes the model hierarchy as part of the target application.

 Off

Excludes the model hierarchy from the target application.

Tips

Including the model hierarchy in the target application:

- Lets you connect to the target computer from xPC Target Explorer without being in the target application build directory.
- Can increase the size of the target application, depending on the size of the model.

Command-Line Information

Parameter: `xPCGenerateXML`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Monitor Signals with xPC Target Explorer"

Enable Stateflow animation

Enables visualization of Stateflow® chart animation.

Settings

Default: off

 On

Enables visualization of Stateflow chart animation.

 Off

Disables visualization of Stateflow chart animation.

Command-Line Information

Parameter: `xPCEnableSFAnimation`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

See Also

"Animate Stateflow Charts with Simulink External Mode"

6

Target Computer Command-Line Interface Reference

Target Computer Commands

In this section...

Introduction

You have a limited set of commands that you can use to work the target application after it has been loaded to the target computer, and to interface with the scopes for that application.

The target computer command-line interface enables you to work with target and scope objects in a limited capacity. Methods let you interact directly with the scope or target. Property commands let you work with target and scope properties. Variable commands let you alias target computer command-line interface commands to names of your choice.

Refer to "Target Computer Command-Line Interface" for a description of how to use these methods and commands.

Target Object Methods

When you are using the target computer command-line interface, target object methods are limited to starting and stopping the target application.

The following table lists the syntax for the target commands that you can use on the target computer. The equivalent MATLAB syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods. These methods assume that you have already loaded the target application onto the target computer.

Target Computer Command	Description and Syntax	MATLAB Equivalent
`start`	Start the target application currently loaded on the target computer. Syntax: `start`	`tg.start` or `+tg`
`stop`	Stop the target application currently running on the target computer. Syntax: `stop`	`tg.stop` or `-tg`
`reboot`	Reboot the target computer. Syntax: `reboot`	`tg.reboot`

Target Object Property Commands

When you are using the target computer command-line interface, target object properties are limited to parameters, signals, stop time, and sample time. Note the difference between a parameter index (0, 1, . . .) and a parameter name (P0, P1, . . .).

The following table lists the syntax for the target commands that you can use to manipulate target object properties. The MATLAB equivalent syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods.

Target Computer Command	Description and Syntax	MATLAB Equivalent
`getpar`	Display the value of a block parameter using the parameter index. Syntax: `getpar parameter_index`	`get(tg, 'parameter_name')`
`setpar`	Change the value of a block parameter using the parameter index. Syntax: `setpar parameter_index = floating_point_number`	`set(tg, 'parameter_name', number)`
`stoptime`	Enter a new stop time. Use `inf` to run the target application until you manually stop it or reset the target computer. Syntax: `stoptime = floating_point_number`	`tg.stoptime = number`
`sampletime`	Enter a new sample time. Syntax: `sampletime = floating_point_number`	`tg.sampletime = number` `set(tg, 'SampleTime', number)`

Target Computer Command	Description and Syntax	MATLAB Equivalent
`P#`	Display or change the value of a block parameter. For example, `P2` or `P2=1.23e-4`. Syntax: `parameter_name` or `parameter_name = floating_point_number` `parameter_name` is `P0, P1, . . .`	`tg.getparam(parameter_index)` `tg.setparam(parameter_index,floating_point_number)`
`S#`	Display the value of a signal. For example, `S2`. Syntax: `signal_name` `signal_name` is `S0, S1, . . .`	`tg.getsignal(signal_index)`

Scope Object Methods

When using the target computer command-line interface, you use scope object methods to start a scope and add signal traces. Notice that the methods `addscope` and `remscope` are target object methods on the host PC, and notice the difference between a signal index (`0`, `1`, . . .) and a signal name (`S0`, `S1`, . . .).

The following table lists the syntax for the target commands that you can use on the target computer. The MATLAB equivalent syntax is shown in the right column. The target object name `tg` and the scope object name `sc` are used as an example for the MATLAB methods.

Target Computer Command	Description and Syntax	MATLAB Equivalent
`addscope`	`addscope scope_index` `addscope`	`tg.addscope(scope_index)` `tg.addscope`
`remscope`	`remscope scope_index` `remscope all`	`tg.remscope(scope_index)` `tg.remscope`
`startscope`	`startscope scope_index`	`sc.start` or `+sc`
`stopscope`	`stopscope scope_index`	`sc.stop` or `-sc`
`addsignal`	`addsignal scope_index = signal_index1, signal_index2, . . .`	`sc.addsignal(signal_index_vector)`
`remsignal`	`remsignal scope_index = signal_index1, signal_index2, . . .`	`sc.remsignal(signal_index_vector)`
`viewmode`	Zoom in to one scope or zoom out to all scopes. Syntax: `viewmode scope_index` or left-click the scope window `viewmode 'all'` or right-click any scope window Press the function key for the scope, and then press **V** to toggle `viewmode`.	`tg.viewMode = scope_index` `tg.viewMode = 'all'`

`ylimit`	`ylimit scope_index`
	`ylimit scope_index = auto`
	`ylimit scope_index = num1, num2`
		`sc.YLimit`
		`sc.YLimit='auto'`
		`sc.YLimit([num1 num2])`

`grid`	`grid scope_index on` `grid scope_index off`	`sc.Grid = on` `sc.Grid = off`

Scope Object Property Commands

When you use the target computer command-line interface, scope object properties are limited to those shown in the following table. Notice the difference between a scope index (`0`, `1`, . . .) and the MATLAB variable name for the scope object on the host PC. The scope index is indicated in the top left corner of a scope window (`SC0`, `SC1`, . . .).

If a scope is running, you need to stop the scope before you can change a scope property.

The following table lists the syntax for the target commands that you can use on the target computer. The equivalent MATLAB syntax is shown in the right column, and the scope object name `sc` is used as an example for the MATLAB methods

Target Computer Command	MATLAB Equivalent
`numsamples scope_index = number`	`sc.NumSamples = number`
`decimation scope_index= number`	`sc.Decimation = number`
`scopemode scope_index = 0 or numerical, 1 or redraw, 2 or sliding, 3 or rolling`	`sc.Mode = 'numerical', 'redraw', 'sliding', 'rolling'`
`triggermode scope_index = 0, freerun, 1, software, 2, signal, 3, scope`	`sc.TriggerMode = 'freerun', 'software', 'signal', 'scope'`
`numprepostsamples scope_index = number`	`sc.NumPrePostSamples = number`
`triggersignal scope_index = signal_index`	`sc.TriggerSignal = signal_index`
`triggersample scope_index = number`	`sc.TriggerSample = number`
`triggerlevel scope_index = number`	`sc.TriggerLevel = number`
`triggerslope scope_index = 0, either, 1, rising, 2, falling`	`sc.TriggerSlope = 'Either', 'Rising', 'Falling'`

Target Computer Command	MATLAB Equivalent
`triggerscope scope_index2 = scope_index1`	`sc.TriggerScope = scope_index1`
`triggerscopesample scope_index= integer`	`sc.TriggerScopeSample = integer`
Press the function key for the scope, and then press **S**.	`sc.trigger`

Aliasing with Variable Commands

The following table lists the syntax for the aliasing variable commands that you can use on the target computer. The MATLAB equivalent syntax is shown in the right column.

Target Computer Command	Description and Syntax	MATLAB Equivalent
`setvar`	Set a variable to a value. Later you can use that variable to do a macro expansion. Syntax: `setvar variable_name = target_pc_command` For example, you can type `setvar aa=startscope 2`, `setvar bb=stopscope 2`.	None
`getvar`	Display the value of a variable. Syntax: `getvar variable_name`	None
`delvar`	Delete a variable. Syntax: `delvar variable_name`	None
`delallvar`	Delete all variables. Syntax: `delallvar`	None
`showvar`	Display a list of variables. Syntax: `showvar`	None