

Simulink<sup>®</sup> Real-Time<sup>™</sup>

Reference



MATLAB<sup>®</sup>&SIMULINK<sup>®</sup>

R2015a



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*Simulink<sup>®</sup> Real-Time<sup>™</sup> Reference*

© COPYRIGHT 2002–2015 by The MathWorks, Inc.

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

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

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://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](http://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)
March 2013	Online only	Revised for Version 5.4 (Release 2013a)
September 2013	Online only	Revised for Version 5.5 (Release 2013b)
March 2014	Online only	Revised for Version 6.0 (Release 2014a)
October 2014	Online only	Revised for Version 6.1 (Release 2014b)
March 2015	Online only	Revised for Version 6.2 (Release 2015a)



## Functions

1

## Simulink Real-Time API Reference for C

2

## Configuration Parameters

3

<b>Configuration Parameters</b> .....	<b>3-2</b>
Simulink Real-Time Options Pane .....	<b>3-3</b>
Automatically download application after building .....	<b>3-4</b>
Download to default target PC .....	<b>3-5</b>
Specify target PC name .....	<b>3-6</b>
Name of Simulink Real-Time object created by build process ..	<b>3-7</b>
Use default communication timeout .....	<b>3-8</b>
Specify the communication timeout in seconds .....	<b>3-9</b>
Execution mode .....	<b>3-10</b>
Real-time interrupt source .....	<b>3-11</b>
I/O board generating the interrupt .....	<b>3-12</b>
PCI slot (-1: autosearch) or ISA base address .....	<b>3-16</b>
Log Task Execution Time .....	<b>3-17</b>
Signal logging data buffer size in doubles .....	<b>3-18</b>
Number of events (each uses 20 bytes) .....	<b>3-21</b>
Double buffer parameter changes .....	<b>3-22</b>
Load a parameter set from a file on the designated target file system .....	<b>3-23</b>
File name .....	<b>3-24</b>
Generate CANape extensions .....	<b>3-25</b>

Enable Stateflow animation .....	3-26
----------------------------------	------

## TLC Options Parameters

4

## Signal and Parameter Instruments

5

<b>Instruments and Instrument Panels .....</b>	<b>5-2</b>
Instrument Selection .....	5-2
Simulink Real-Time Explorer Within MATLAB and as Standalone Executable .....	5-6
Guidelines for Exporting Instrumented Models .....	5-9
<b>Create Instrument Panel .....</b>	<b>5-12</b>
<b>Configure Instrument for Parameter Tuning .....</b>	<b>5-13</b>
<b>Configure Instruments for Signal Display .....</b>	<b>5-16</b>
<b>Save and Load Instrument Panels .....</b>	<b>5-20</b>
<b>Save and Restore Layouts .....</b>	<b>5-21</b>
<b>Run Instrumented Model .....</b>	<b>5-22</b>
<b>Prepare Deployment Environment for Export .....</b>	<b>5-25</b>
<b>Create Parameter Groups for Export .....</b>	<b>5-27</b>
<b>Create Signal Groups for Export .....</b>	<b>5-29</b>
<b>Prepare Instrument Panel Configuration for Export .....</b>	<b>5-31</b>
<b>Export Instrument Panel Configuration .....</b>	<b>5-33</b>

<b>Deploy and Run Standalone Configuration</b> .....	<b>5-35</b>
<b>Instruments — Alphabetical List</b> .....	<b>5-37</b>

## **Target Computer Command-Line Interface Reference**

# **6**

<b>Target Computer Commands</b> .....	<b>6-2</b>
Target Object Function Commands .....	<b>6-2</b>
Target Object Property Commands .....	<b>6-3</b>
Scope and Video Object Function Commands .....	<b>6-4</b>
Scope Object Property Commands .....	<b>6-6</b>
Aliasing with Variable Commands .....	<b>6-10</b>





# Functions

---

## getxpcenv

List environment properties assigned to MATLAB variable (not recommended)

### Syntax

```
getxpcenv  
getxpcenv propertyname
```

### Description

getxpcenv displays, in the Command Window, the property names and current property values for the Simulink® Real-Time™ environment.


---

**Note:** Function getxpcenv will be removed in a future release. Use SimulinkRealTime.getTargetSettings and Target Settings Properties instead.

---

getxpcenv propertyname displays the current value of property propertyname. The environment properties define communication between the development and target computers and the type of target boot kernel created during the setup process.

To access the environment properties in Simulink Real-Time Explorer:

- 1 In the **Targets** pane, expand a target computer node.
- 2 In the toolbar, click the Target Properties icon .
- 3 Expand the sections **Host-to-Target communication**, **Target settings**, or **Boot configuration**.
  - “Host-to-Target Communication” on page 1-3
  - “Target Settings” on page 1-8
  - “Boot Configuration” on page 1-11
  - “Host Configuration” on page 1-12

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you must set the other properties that start with TcpIp.</p> <hr/> <p><b>Note:</b> RS-232 communication type will be removed in a future release. Use TCP/IP instead.</p>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>

Environment Property	Description
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of Simulink Real-Time 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.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your development 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.</p>
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>

Environment Property	Description
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time 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 development computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p>

Environment Property	Description
<p>TcpIpTargetISAIRQ</p>	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>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.</p>

Environment Property	Description
TcpIpTargetISAMemPort	<p>Property value is '0xnnnn'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>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.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (<b>telnet</b>, <b>ftp</b>, . . .) and is only of use on the target computer.</p>

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <math>n</math> indicates the index number for the Ethernet card on a target computer. Note that the <math>(n - 1)</math>th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>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'.</p>
MaxModelSize	<p>Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported.</p> <p>Select 1 MB or 4 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>Setting <b>Model size</b> is enabled for <b>Boot mode Stand Alone</b> only.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the real-time application. Memory not used by the real-time application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.</p>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p>



Environment Property	Description
	<p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
Name	Target computer name.
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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 slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The development computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, real-time application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the real-time application reads the target computer BIOS and determines the amount of memory up to a maximum of 4 GB. If the real-time application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 4 GB.</p> <p>The Simulink Real-Time kernel can use only 4 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>

Environment Property	Description
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of Simulink Real-Time Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network.</p> <p>Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p> <p>xx:xx:xx:xx:xx:xx</p>

Environment Property	Description
	<p>To update the MAC address in Simulink Real-Time Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> 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.</p>

## Host Configuration

Environment Property	Description
Version	<p>Simulink Real-Time version number. Displayed only from <code>getxpcenv</code> when called without arguments.</p>

## Examples

Display the Simulink Real-Time environment in the format shown below.

```

getxpcenv
Simulink Real-Time Target Settings

    Name                : TargetPC1

    TargetRAMSizeMB     : Auto
    MaxModelSize        : 1MB
    SecondaryIDE        : off
    NonPentiumSupport   : off
    MulticoreSupport    : on
    LegacyMultiCoreConfig : off
    USBSupport          : on
    ShowHardware        : off
    EthernetIndex       : 0

    HostTargetComm      : TcpIp
  
```

```
TcpIpTargetAddress      : 10.10.10.15
TcpIpTargetPort        : 22222
TcpIpSubNetMask        : 255.255.255.0
TcpIpGateway           : 10.10.10.100
RS232HostPort          : COM1
RS232Baudrate          : 115200
TcpIpTargetDriver      : Auto
TcpIpTargetBusType     : PCI
TcpIpTargetISAMemPort  : 0x300
TcpIpTargetISAIRQ      : 5

TargetScope            : Enabled

TargetBoot             : NetworkBoot
TargetMACAddress       : 90:e2:ba:17:5d:15
```

Return specific environment property value.

```
env = getxpcenv('HostTargetComm')
env =

    'TcpIp'
```

## See Also

xpcbootdisk | setxpcenv

## getxpcinfo

Retrieve diagnostic information to help troubleshoot configuration issues (not recommended)

### Syntax

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

### Arguments

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

### Description

`getxpcinfo` returns diagnostic information for troubleshooting Simulink Real-Time 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.

---

**Note:** Function `getxpcinfo` will be removed in a future release. Use `SimulinkRealTime.getSupportInfo` instead.

---

`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.

---

## getxpcpci

Determine PCI boards installed in target computer (not recommended)

### Syntax

```
getxpcpci 'installed'
getxpcpci 'ethernet'
getxpcpci 'all'
getxpcpci 'verbose'

getxpcpci 'supported'
getxpcpci 'supported' 'ethernet'

pci_devices = getxpcpci('installed')
pci_devices = getxpcpci('ethernet')
pci_devices = getxpcpci('all')
pci_devices = getxpcpci('verbose')
pci_devices = getxpcpci(target_object, ___ )

pci_devices_supported = getxpcpci('supported')
pci_devices_supported = getxpcpci('supported', 'ethernet')
```

### Description

getxpcpci 'installed' queries the default target computer for installed PCI devices (boards) that are supported by driver blocks in the Simulink Real-Time block library.

---

**Note:** Function getxpcpci will be removed in a future release. Use SimulinkRealTime.target.getPCIInfo instead.

---

The call displays in the Command Window information about the PCI devices found, including:

- PCI bus number
- Slot number

- Assigned IRQ number
- Vendor (manufacturer) name
- Device (board) name
- Device type
- Vendor PCI ID
- Device PCI ID
- Device release version.

Before you can use this call, you must meet the following preconditions:

- The Ethernet link must be working. Before you can use `getxpcpci`, the function `xpctargetping` must return `success`.
- Either a real-time application is loaded or the loader is active. Before building the model, you can use `getxpcpci` to find resources to enter into a driver block dialog box. Such resources include PCI bus number, slot number, and assigned IRQ number.

`getxpcpci 'ethernet'` queries the default target computer for installed Ethernet controllers supported by Simulink Real-Time.

`getxpcpci 'all'` displays information about all of the PCI devices found on the default target computer. This information includes graphics controllers, network cards, SCSI cards, and devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

`getxpcpci 'verbose'` shows the information displayed by `getxpcpci 'all'` for the default target computer, plus information about the PCI addresses assigned to this board by the BIOS.

`getxpcpci 'supported'` displays a list of the PCI devices currently supported by the Simulink Real-Time block library. This call does not access the target computer, so the Ethernet link does not have to be active.

`getxpcpci 'supported' 'ethernet'` displays a list of the Ethernet controllers that are supported by Simulink Real-Time. This call does not access the target computer, so the Ethernet link does not have to be active.

`pci_devices = getxpcpci('installed')` queries the default target computer for installed PCI devices (boards) that are supported by driver blocks in the Simulink Real-Time block library. The call returns a structure containing information about the PCI devices found on the target computer.



`pci_devices = getxpcpci('ethernet')` queries the default target computer for installed Ethernet controllers that are supported by Simulink Real-Time. The call returns a structure containing information about the Ethernet controllers found on the target computer.

`pci_devices = getxpcpci('all')` returns a structure containing information about all PCI devices found on the default target computer. This structure includes information about the PCI addresses assigned to this board by the BIOS.

`pci_devices = getxpcpci('verbose')` returns a structure containing information about all PCI devices found on the default target computer. This structure includes information about the PCI addresses assigned to this board by the BIOS.

`pci_devices = getxpcpci(target_object, ___)` applies the option arguments to the target computer represented by `target_object`.

`pci_devices_supported = getxpcpci('supported')` returns a structure containing a list of PCI devices currently supported by the Simulink Real-Time block library. This call does not access the target computer, so the Ethernet link does not have to be active.

`pci_devices_supported = getxpcpci('supported','ethernet')` returns a structure containing a list of the Ethernet controllers supported by Simulink Real-Time. This call does not access the target computer, so the Ethernet link does not have to be active.

## Examples

### Display information for PCI devices on default computer that the Simulink Real-Time block library supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer.

```
xpctargetping
```

```
getxpcpci 'installed'
```

```
List of installed PCI devices:
```

```
Measurement Computing    PCI-DI024
  Bus 1, Slot 11, IRQ 10
  DI DO
  VendorID 0x1307, DeviceID 0x0028,
    SubVendorID 0x1307, SubDeviceID 0x0028
  A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
  Released in: R14SP2 or Earlier
```

```
.
.
.
```

## Display information for Ethernet controllers on default computer that Simulink Real-Time supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer.

```
xpctargetping
```

```
getxpcpci 'ethernet'
```

```
List of installed PCI devices:
```

```
Intel                82541GI_LF
  Bus 16, Slot 4, IRQ 10
  Ethernet controller
  VendorID 0x8086, DeviceID 0x107c, SubVendorID 0x8086,
    SubDeviceID 0x1376
  Released in: R2006b
  Notes: Intel Gigabit Ethernet series
```

## Display information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer.

```
xpctargetping
```

```
getxpcpci 'all'
```

```
List of installed PCI devices:
```

```

Intel                               Unknown
  Bus 0, Slot 0, IRQ 0
  Host Bridge
  VendorID 0x8086, DeviceID 0x1130,
    SubVendorID 0x8086, SubDeviceID 0x4532
.
.
.
Measurement Computing             PCI-DIO24
  Bus 1, Slot 11, IRQ 10
  DI DO
  VendorID 0x1307, DeviceID 0x0028,
    SubVendorID 0x1307, SubDeviceID 0x0028
  A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
  Released in: R14SP2 or Earlier
.
.
.

```

### Display verbose information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer.

```
xpctargetping
```

```
getxpcpci 'verbose'
```

```
List of installed PCI devices:
```

```

Intel                               Unknown
  Bus 0, Slot 0, IRQ 0
  Host Bridge
  VendorID 0x8086, DeviceID 0x1130,
    SubVendorID 0x8086, SubDeviceID 0x4532
  BaseClass 6, SubClass 0
  BAR BaseAddress AddressSpace  MemoryType PreFetchable
    0)   E8000000      Memory    32-bit decoder    no
.
.
.
Measurement Computing             PCI-DIO24
  Bus 1, Slot 11, IRQ 10
  DI DO

```

```

VendorID 0x1307, DeviceID 0x0028,
  SubVendorID 0x1307, SubDeviceID 0x0028
A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
Released in: R14SP2 or Earlier
BaseClass FF, SubClass FF
BAR BaseAddress AddressSpace
  1)          DC00          I/O
  2)          DFF4          I/O
.
.
.

```

**Display information for all PCI devices that the Simulink Real-Time block library supports**

At the MATLAB prompt, type the command on the development computer.

```
getxpcpci 'supported'
```

List of supported PCI devices:

Vendor	Device	Type...
ADLINK	PCI-6208A	AO DI DO...
B&B Electronics (Quatech)	DSCP-200/300 (PXI)	Serial Ports...
.	.	.
Speedgoat	I0321 (PMC-FPGA)	AI (I0321-5)...
Speedgoat	I0331 (PMC-FPGA)	DI DO (LVDS/LVCMOS)...

**Display information for all Ethernet controllers that Simulink Real-Time supports**

At the MATLAB prompt, type the command on the development computer.

```
getxpcpci 'supported' 'ethernet'
```

List of supported Ethernet controllers:

Vendor	Device	VendorID	DeviceID	Release
3Com	3c900B Combo	10B7	9005	R2006a+
3Com	3c905B Combo	10B7	9058	R2006a+
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Winbond Electronics 89C940	1050	5A5A	R2006a+
Winbond Electronics 89C940	8C4A	1980	R2006a+

### Return information for PCI devices on default computer that the Simulink Real-Time block library supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer. Display the first structure in the vector.

```
xpctargetping
```

```
pci_devices = getxpcpci('installed');
pci_devices(1)
```

```
ans =
```

```

        Bus: 1
        Slot: 11
        VendorID: '1307'
        DeviceID: '28'
        SubVendorID: '1307'
        SubDeviceID: '28'
        BaseClass: 'FF'
        SubClass: 'FF'
        Interrupt: 10
BaseAddresses: [1x6 struct]
        VendorName: 'Measurement Computing'
        Release: 'R14SP2 or Earlier'
        Notes: ''
        DeviceName: 'PCI-DI024'
        DeviceType: 'DI D0'
        ADChan: '0'
        DACHan: '0'
        DIOChan: '24'
```

### Return information for Ethernet controllers on default computer that Simulink Real-Time supports

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer. Display the first structure in the vector.

```
xpctargetping

pci_devices = getxpcpci('ethernet');
pci_devices(1)

ans =

        Bus: 16
        Slot: 4
    VendorID: '8086'
    DeviceID: '107C'
SubVendorID: '8086'
SubDeviceID: '1376'
    BaseClass: '2'
        SubClass: '0'
    Interrupt: 10
BaseAddresses: [1x6 struct]
    VendorName: 'Intel'
        Release: 'R2006b'
        Notes: 'Intel Gigabit Ethernet series'
    DeviceName: '82541GI_LF'
    DeviceType: 'Ethernet controller'
        ADChan: ''
        DACHan: ''
        DIOChan: ''
```

### **Return information for all PCI devices on default computer**

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer. Display the first structure in the vector.

```
xpctargetping

pci_devices = getxpcpci('all');
pci_devices(1)

ans =

        Bus: 0
        Slot: 0
    VendorID: '8086'
    DeviceID: '1130'
SubVendorID: '8086'
```

```

SubDeviceID: '4532'
BaseClass: '6'
SubClass: '0'
Interrupt: 0
BaseAddresses: [1x6 struct]
VendorName: 'Intel'
Release: ''
Notes: ''
DeviceName: 'Unknown'
DeviceType: 'Host Bridge'
ADChan: ''
DACHan: ''
DIOChan: ''

```

### Return verbose information for all PCI devices on default computer

Start the default target computer with the Simulink Real-Time kernel. Verify the connection between the host and the target computer. At the MATLAB command prompt, type the command on the development computer. Display the first structure in the vector.

```
xpctargetping
```

```
pci_devices = getxpcpci('verbose');
pci_devices(1)
```

```
ans =
```

```

Bus: 0
Slot: 0
VendorID: '8086'
DeviceID: '1130'
SubVendorID: '8086'
SubDeviceID: '4532'
BaseClass: '6'
SubClass: '0'
Interrupt: 0
BaseAddresses: [1x6 struct]
VendorName: 'Intel'
Release: ''
Notes: ''
DeviceName: 'Unknown'
DeviceType: 'Host Bridge'
ADChan: ''
DACHan: ''

```

```
DIOChan: ''
```

### Return verbose information for all PCI devices via target\_object

Start the default target computer with the Simulink Real-Time kernel. Get the `target_object` using `xpctarget.xpc`. Verify the connection between the host and the target computer. At the MATLAB prompt, type the command on the development computer. Display the first structure in the vector.

```
target_object = xpctarget.xpc('myTargetPC');
target_object.targetping

pci_devices = getxpcpci(target_object, 'verbose');
pci_devices(1)

ans =
```

```
        Bus: 0
        Slot: 0
    VendorID: '8086'
    DeviceID: '1130'
SubVendorID: '8086'
SubDeviceID: '4532'
    BaseClass: '6'
    SubClass: '0'
    Interrupt: 0
BaseAddresses: [1x6 struct]
    VendorName: 'Intel'
        Release: ''
        Notes: ''
    DeviceName: 'Unknown'
    DeviceType: 'Host Bridge'
        ADChan: ''
        DAChan: ''
        DIOChan: ''
```

### Return information for all PCI devices that the Simulink Real-Time block library supports

At the MATLAB prompt, type the command on the development computer.

```
pci_devices_supported = getxpcpci('supported');
pci_devices_supported(1)

ans =
```



```

VendorID: '144A'
DeviceID: '6208'
SubVendorID: '-1'
SubDeviceID: '-1'
DeviceName: 'PCI-6208A'
VendorName: 'ADLINK'
DeviceType: 'AO DI DO'
DChan: '8'
ADChan: '0'
DIOChan: '4'
Release: 'R14SP2 or Earlier'
Notes: 'PCI-6208A features 8 current outputs w...'

```

### Return information for all Ethernet controllers that Simulink Real-Time supports

At the MATLAB prompt, type the command on the development computer.

```
pci_devices_supported = getxpcpci('supported', 'ethernet');
pci_devices_supported(1)
```

ans =

```

VendorID: '10B7'
DeviceID: '9005'
SubVendorID: '-1'
SubDeviceID: '-1'
DeviceName: '3c900B Combo'
VendorName: '3Com'
DeviceType: 'Ethernet controller'
DChan: ''
ADChan: ''
DIOChan: ''
Release: 'R2006a+'
Notes: '3Com Etherlink 90x series'

```

- “Where to Find PCI Board Information”
- “Command-Line Ethernet Card Selection by Index”

## Input Arguments

**target\_object** — Object representing target computer  
object created by xpctarget.xpc

Object representing the target computer being queried, as returned by `xpctarget.xpc`.

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

Data Types: `function_handle`

## Output Arguments

### **pci\_devices** — Information about the PCI devices in the target computer

vector

The vector returned by `getxpcpci` without an argument contains information only for those PCI devices supported by Simulink Real-Time blocks. The vectors returned by `getxpcpci` with the arguments `'all'` and `'verbose'` contain information about all PCI devices in the target computer and are identical.

The fields in this structure are:

#### **Bus** — PCI bus where device resides

scalar

`Bus` and `Slot` are used together to uniquely identify the location of a device or bus adapter in the target computer.

#### **Slot** — PCI slot where device resides

scalar

`Slot` and `Bus` are used together to uniquely identify the location of a device or bus adapter in the target computer.

#### **VendorID** — Identifier for manufacturer of the device

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

#### **DeviceID** — Identifier for device among those manufactured by the vendor

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this device or bus adapter.

**SubVendorID — Identifier for manufacturer of subsystem**

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

**SubDeviceID — Identifier for subsystem among those manufactured by the subvendor**

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this subsystem (board).

**BaseClass — Standard PCI class of the device**

string

Hexadecimal numeric string containing the standard PCI base classification of this device or bus adapter. **BaseClass** and **SubClass** together identify the type and function of the device.

**SubClass — Standard PCI subclass of the device**

string

Hexadecimal numeric string containing the standard PCI subclass classification of this device or bus adapter. **SubClass** and **BaseClass** together identify the type and function of the device.

**Interrupt — IRQ used by the device**

scalar

Provides the board-level interrupt used by the device or bus adapter to trigger I/O with the target computer CPU.

**BaseAddresses — Information for each Base Address Register (BAR) used by the device**

vector

For each BAR used by this device or bus adapter, the vector contains a structure with the following fields:

**AddressSpaceIndicator — Indicates whether the address is a memory or I/O address**

0 | 1

- 0 — Address is memory address

- 1 — Address is I/O address

**BaseAddress — Memory address used by the device**

string

Hexadecimal string containing the base memory address used by the device.

**MemoryType — Indicates the size of the address decode, 32-bit or 64-bit**

0 | 1

Not used if `AddressSpaceIndicator` is 1 (I/O address).

- 0 — 32-bit address decode
- 1 — 64-bit address decode

**Prefetchable — Indicates whether the memory is prefetchable**

0 | 1

Not used if `AddressSpaceIndicator` is 1 (I/O address).

- 0 — Address not prefetchable
- 1 — Address prefetchable

**VendorName — Name of vendor of device**

string

Identifies the vendor of the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

**Release — MATLAB® release version in which driver became available**

string

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

**Notes — Additional information about the device**

string

Contains additional description of the device or bus adapter.

**DeviceName — Name of device**

string

Identifies the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

**DeviceType — Identifies the functions of the device**

string

Contains abbreviations such as 'DI' (digital input) that indicate the function or functions of the device or bus adapter.

**ADChan — Number of analog inputs**

string

Decimal numeric string containing the number of analog inputs to the device.

**DACHan — Number of analog outputs**

string

Decimal numeric string containing the number of analog outputs from the device.

**DIOChan — Number of digital inputs and outputs**

string

Decimal numeric string containing the number of digital inputs and outputs to and from the device.

**pci\_devices\_supported — Information about the PCI devices supported by the product**

vector

Vector of information about the devices and bus adapters represented by blocks in the Simulink Real-Time block library.

The fields are as follows:

**VendorID — Identifier for manufacturer of the device**

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of this device or bus adapter.

**DeviceID — Identifier for device among those manufactured by the vendor**

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this device or bus adapter.

**SubVendorID — Identifier for manufacturer of subsystem**

string

Hexadecimal numeric string containing the identifier that the PCI standards organization assigns to the manufacturer of the entire subsystem (board).

**SubDeviceID — Identifier for subsystem among those manufactured by the subvendor**

string

Hexadecimal numeric string containing the identifier that the manufacturer assigns to this subsystem (board).

**DeviceName — Name of device**

string

Identifies the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

**VendorName — Name of vendor of device**

string

Identifies the vendor of the specific device or bus adapter. Set to 'Unknown' for unknown devices or bus adapters.

**DeviceType — Identifies the functions of the device**

string

Contains abbreviations such as 'DI' (digital input) that indicate the function or functions of the device or bus adapter.

**DACHan — Number of analog outputs**

string

Decimal numeric string containing the number of analog outputs from the device.

**ADChan — Number of analog inputs**

string

Decimal numeric string containing the number of analog inputs to the device.

**DIOChan — Number of digital inputs and outputs**

string

Decimal numeric string containing the number of digital inputs and outputs to and from the device.

**Release — MATLAB release version in which driver became available**

string

If the Simulink Real-Time block library supports the device, it contains the MATLAB and Simulink release version in which the driver was released. Otherwise, it contains an empty vector.

**Notes — Additional information about the device**

string

Contains additional description of the device or bus adapter.

## More About

- “PCI Bus I/O Devices”

## readxpcfile

Read real-time Scope file format data (not recommended)

### Syntax

```
matlab_data = readxpcfile(xpcfile_name)
matlab_data = readxpcfile(xpcfile_data)
```

### Description

`matlab_data = readxpcfile(xpcfile_name)` takes as an argument the name of a development computer file containing a vector of byte data (`uint8`). The file is copied from the target computer using `xpctarget.ftp` Class methods.

---

**Note:** Function `readxpcfile` will be removed in a future release. Use `SimulinkRealTime.utils.getFileScopeData` instead.

---

`matlab_data = readxpcfile(xpcfile_data)` takes as an argument a MATLAB variable containing a vector of byte data (`uint8`). The data is read from the target computer using `xpctarget.fs` Class methods.

### Examples

#### Using `xpcfile_name` argument to read file and plot results

Upload file 'data.dat' using `xpctarget.ftp` Class methods. Read the file on the host using `readxpcfile`. Plot the results.

Upload file 'data.dat' from the target computer.

```
xpcftp = xpctarget.ftp;
get(xpcftp, 'data.dat')
```

Read the file and process its data into MATLAB format.



```
matlab_data = readxpcfile('data.dat');
```

Plot the signal data (column 1) on the Y axis against time (column 2) on the X axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))
xlabel(matlab_data.signalNames(2))
ylabel(matlab_data.signalNames(1))
```

### Using xpcfile\_data argument to store data, convert to MATLAB format, and plot results

Read file 'data.dat' on the target computer from the host. Store the data in a MATLAB workspace variable. Convert the data to MATLAB format using `readxpcfile`. Plot the results.

Read file 'data.dat' from the target computer.

```
f = xpctarget.fs;
h = fopen(f, 'data.dat');
xpcfile_data = fread(f, h);
fclose(f, h)
```

Process data from the workspace variable into MATLAB format.

```
matlab_data = readxpcfile(xpcfile_data);
```

Plot the signal data (column 1) on the Y axis against time (column 2) on the X axis.

```
plot(matlab_data.data(:,2), matlab_data.data(:,1))
xlabel(matlab_data.signalNames(2))
ylabel(matlab_data.signalNames(1))
```

## Input Arguments

**xpcfile\_name** — Name of file from which to read real-time Scope file format data  
'data.dat'

File must contain a vector of `uint8` data.

Data Types: `char`

**xpcfile\_data** — Workspace variable containing real-time Scope file format data  
vector

Data Types: `uint8`

## Output Arguments

### **matlab\_data** — State and time data for plotting

structure

The state and time data is stored in a structure containing six fields. The key fields are `numSignals`, `data`, and `signalNames`.

### **version** — Version code

0 (default) | double

Internal

### **sector** — Sector of data file

0 (default) | double

Internal

### **headersize** — Number of bytes of data file header

512 (default) | double

Internal

### **numSignals** — Number of columns containing signal and time data

double

If  $N$  signals are connected to the real-time Scope block,  $\text{numSignals} = N + 1$ .

### **data** — Columns containing signal and time data

double array

The data array contains `numSignals` columns. The first  $N$  columns represent signal state data. The last column contains the time at which the state data is captured.

The data array contains as many rows as there are data points.

### **signalNames** — Names of columns containing signal and time data

cell vector

The `signalNames` vector contains `numSignals` elements. The first  $N$  elements are signal names. The last element is the string `Time`.

## **See Also**

Scope | xpctarget.fs Class | xpctarget.ftp Class

## setxpcenv

Change Simulink Real-Time environment properties (not recommended)

### Syntax

```
setxpcenv  
setxpcenv('property_name', 'property_value')  
setxpcenv('prop_name1', 'prop_value1', 'prop_name2', . . .)
```

### Arguments

<code>property_name</code>	Not case sensitive. Property names can be shortened as long as they can be differentiated from the other property names.
<code>property_value</code>	Character string. Type <code>setxpcenv</code> 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. `setxpcenv` works similarly to the `set` function of the MATLAB Handle Graphics® system.

---

**Note:** Command `setxpcenv` will be removed in a future release. Use Target Settings Properties instead.

---


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

`setxpcenv('property_name', 'property_value')` sets property `property_name` to `property_value`.

`setxpcenv('prop_name1', 'prop_value1', 'prop_name2', . . .)` is called with one or more argument pairs. The first argument of a pair is the property name; the second is the new value for this property.

The environment properties define communication between the development and target computers 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 Simulink Real-Time Explorer window, accessed via the `xpcexplr` function. An understanding of the environment properties will help you configure the Simulink Real-Time environment.

To access the environment properties in Simulink Real-Time Explorer:

- 1 In the **Targets** pane, expand a target computer node.
- 2 In the toolbar, click the Target Properties icon .
- 3 Expand the sections **Host-to-Target communication**, **Target settings**, or **Boot configuration**.
  - “Host-to-Target Communication” on page 1-37
  - “Target Settings” on page 1-43
  - “Boot Configuration” on page 1-46

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you select RS-232, you also must set the property <code>RS232HostPort</code>. If you select TCP/IP, then you must set the other properties that start with <code>TcpIp</code>.</p> <hr/> <p><b>Note:</b> RS-232 communication type will be removed in a future release. Use TCP/IP instead.</p>

Environment Property	Description
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the <b>HostTargetComm</b> property to RS232.</p>
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of Simulink Real-Time 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.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your development 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.</p>

Environment Property	Description
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>

Environment Property	Description
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time 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 development computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p>



Environment Property	Description
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>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.</p>

Environment Property	Description
TcpIpTargetISAMemPort	<p>Property value is '0xnnnn'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>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.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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.</p>

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <math>n</math> indicates the index number for the Ethernet card on a target computer. Note that the <math>(n - 1)</math>th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>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'.</p>
MaxModelSize	<p>Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported.</p> <p>Select 1 MB or 4 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>Setting <b>Model size</b> is enabled for <b>Boot mode Stand Alone</b> only.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the real-time application. Memory not used by the real-time application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.</p>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p>

Environment Property	Description
	<p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
Name	Target computer name.
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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 slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The development computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, real-time application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the real-time application reads the target computer BIOS and determines the amount of memory up to a maximum of 4 GB. If the real-time application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 4 GB.</p> <p>The Simulink Real-Time kernel can use only 4 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>

Environment Property	Description
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of Simulink Real-Time Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p>
TargetMACAddress	<p>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:</p> <p>xx:xx:xx:xx:xx:xx</p>

Environment Property	Description
	To update the MAC address in Simulink Real-Time Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> 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 port of the development computer to COM2.

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

## More About

- “Ethernet Link Setup”
- “Serial Link Setup”
- “Target Boot Methods”
- “Command-Line Setup”

## See Also

getxpcenv | xpcbootdisk

# xpcbench

Benchmark Simulink Real-Time models on target computer

## Syntax

```
xpcbench
xpcbench benchmark
xpcbench benchmark -reboot
xpcbench benchmark -cleanup
xpcbench benchmark -verbose
xpcbench benchmark -reboot -cleanup -verbose
```

```
expected_results = xpcbench()
current_results = xpcbench(benchmark, ___ )
```

## Description

`xpcbench` benchmarks the real-time execution performance of Simulink Real-Time applications on your target computer. It compares the result to stored benchmark results from other computers.

---

**Note:** Function `xpcbench` will be removed in a future release. Use `slrtbench` instead.

---

Benchmark execution includes generating benchmark models, building and downloading Simulink Real-Time applications, searching for the minimal achievable sample time, and displaying results.

`xpcbench` without an argument displays representative results for benchmarks run on various target computers with various compiler versions. Display includes:

- Relative Performance — Bar graph containing the computers tested, ranked by relative performance.
- Minimal achievable sample times in  $\mu\text{s}$  — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.



- **Target Information** — Technical information about the target computers benchmarked.

Depending upon the value of `benchmark`, `xpcbench benchmark` produces different outputs:

- `xpcbench this` displays benchmark results your target computer, compared with the representative benchmark results for other target computers:
  - **Relative Performance** — Bar graph containing the computers tested, ranked by relative performance.
  - **Minimal achievable sample times in  $\mu\text{s}$**  — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.
  - **Target Information** — Technical information about the target computers benchmarked.

The entry for your target computer is highlighted.

- `xpcbench benchmark` prints the benchmark name, the number of blocks, the model build time in seconds, the execution time in seconds, and the minimal achievable sample time in microseconds in the Command Window.

`xpcbench benchmark -reboot` runs the benchmark, then restarts the target computer.

`xpcbench benchmark -cleanup` runs the benchmark, plots or prints benchmark results, and deletes the build files.

`xpcbench benchmark -verbose` prints build output, runs the benchmark, and plots or prints benchmark results.

`xpcbench benchmark -reboot -cleanup -verbose` prints build output, restarts the target computer, deletes build files, and plots or prints results.

You can add zero or more of these control arguments in arbitrary order.

`expected_results = xpcbench()` returns the benchmark results for the five predefined benchmarks in a structure array.

Depending upon the value of `benchmark`, `current_results = xpcbench(benchmark, ___)` returns different results:

- `xpcbench('this')` returns the benchmark results for the predefined benchmarks in a structure array.
- `xpcbench(benchmark)` returns the benchmark results for the specified model in a structure.

## Examples

### `xpcbench`

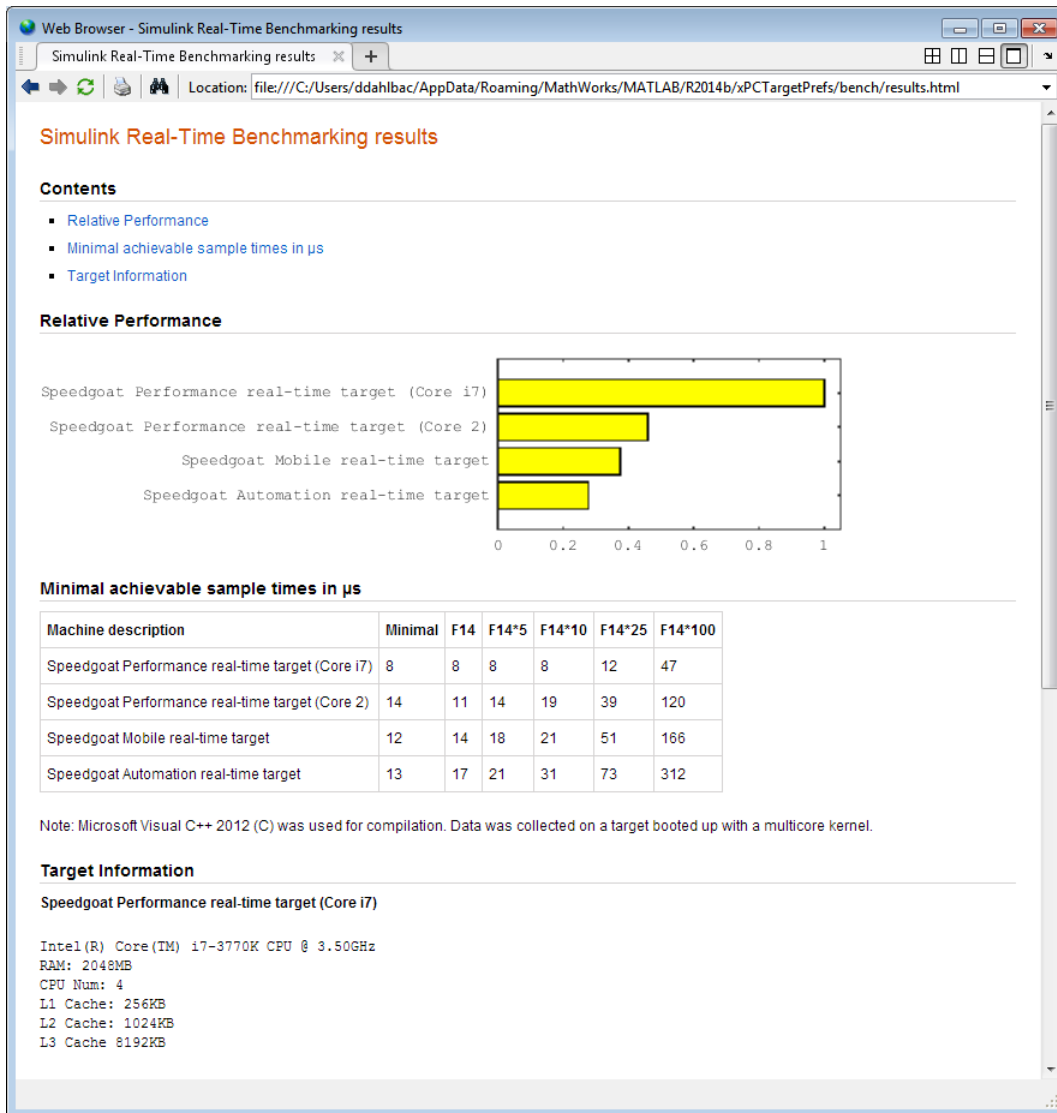
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

`slrttest`

Display representative results on predefined benchmarks.

`xpcbench`



### xpcbench this

Benchmark the target computer with the predefined benchmarks.

Start the target computer and run confidence test.

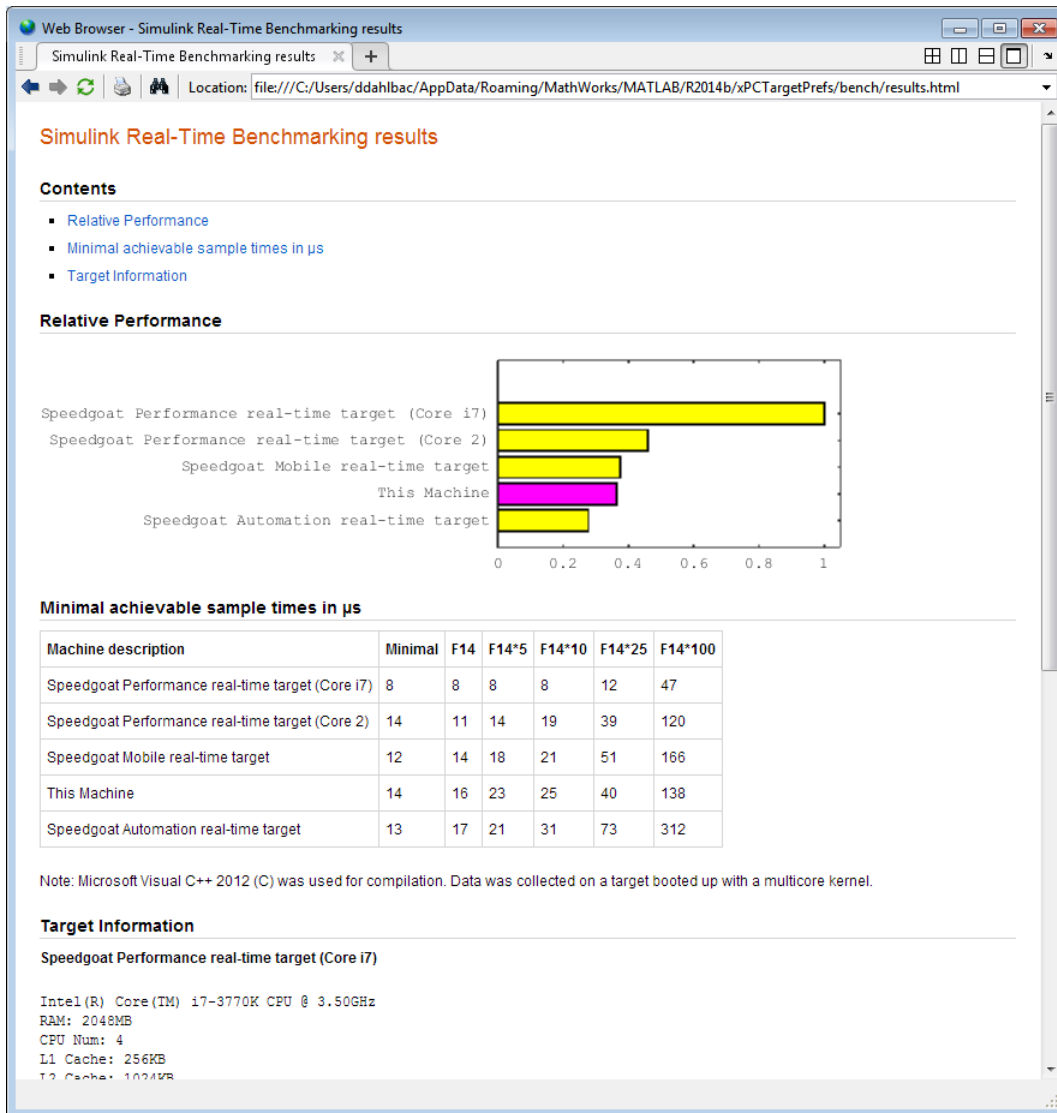
slrttest

Run the benchmark models and display results.

xpcbench `this`

```
### Starting Simulink Real-Time build procedure
    for model: xpcminimal
### Successful completion of build procedure for model: xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal
.
.
.
### Running benchmark for model: f14tmp1
.
.
.
### Running benchmark for model: f14tmp5
.
.
.
### Running benchmark for model: f14tmp10
.
.
.
### Running benchmark for model: f14tmp25
.
.
.
### Running benchmark for model: f14tmp100
```



**xpcbench this -verbose -reboot -cleanup**

Benchmark the target computer with the predefined benchmarks, and then delete build files.

Start the target computer and run confidence test.

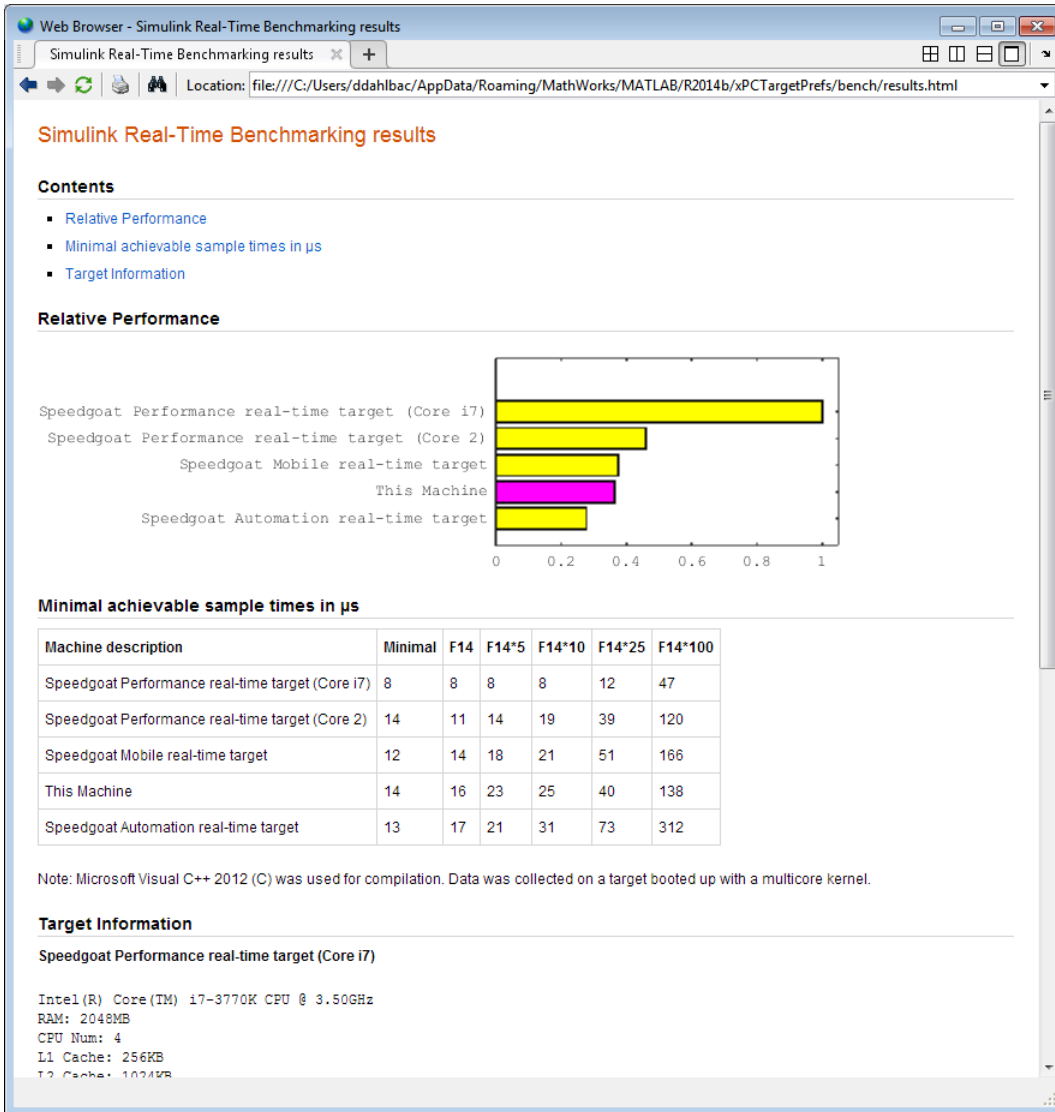
```
slrttest
```

Run the benchmark models, delete build files, and display results.

```
xpcbench this -verbose -reboot -cleanup
```

```
### Starting Simulink Real-Time build procedure
      for model: xpcminimal
### Generating code into build folder: xpcminimal_xpc_rtw
### Invoking Target Language Compiler on xpcminimal.rtw
.
.
.
### Successful completion of build procedure for model:
      xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
      Connected          = Yes
.
.
.
### Running benchmark for model: xpcminimal
### Reboot target: TargetPC1..... OK.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1..... OK.
.
.
.
### Running benchmark for model: f14tmp5
### Reboot target: TargetPC1..... OK.
.
.
.
### Running benchmark for model: f14tmp10
### Reboot target: TargetPC1..... OK.
.
.
.
### Running benchmark for model: f14tmp25
```

```
### Reboot target: TargetPC1..... OK.  
.  
.  
.  
### Running benchmark for model: f14tmp100  
### Reboot target: TargetPC1..... OK.
```



## xpcbench xpcosc

Use model `xpcosc` to benchmark the target computer, then clean up build files

Start the target computer and run confidence test.



slrttest

Run benchmark on xpcosc, delete build files, and print results.

xpcbench xpcosc

```
### Starting Simulink Real-Time build procedure for model: xpcosc
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

```
### Running benchmark for model: xpcosc
```

```
Benchmark results for model:          xpcosc
Number of blocks in model:            10
Elapsed time for model build (sec):    33.4
Elapsed time for model benchmark (sec): 236.7
Minimal achievable sample time (microsec): 12.4
```

**xpcbench xpcosc --verbose -reboot -cleanup**

Use model xpcosc to benchmark the target computer, then clean up build files

Start the target computer and run confidence test.

slrttest

Run benchmark on xpcosc, delete build files, and print results.

xpcbench xpcosc -verbose -reboot -cleanup

```
### Starting Simulink Real-Time build procedure for model: xpcosc
### Generating code into build folder: xpcosc_slrt_rtw
### Invoking Target Language Compiler on xpcosc.rtw
```

```
.
.
.
```

```
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
    Connected          = Yes
```

```
.
```

```
.  
.  
  
### Running benchmark for model: xpcosc  
### Reboot target: TargetPC1..... OK  
  
Benchmark results for model:           xpcosc  
Number of blocks in model:           10  
Elapsed time for model build (sec):   29.4  
Elapsed time for model benchmark (sec): 210.5  
Minimal achievable sample time (microsec): 10.9
```

### **expected\_results = xpcbench()**

Return a structure array containing benchmark results showing what to expect of various target computers.

Start the target computer and run confidence test.

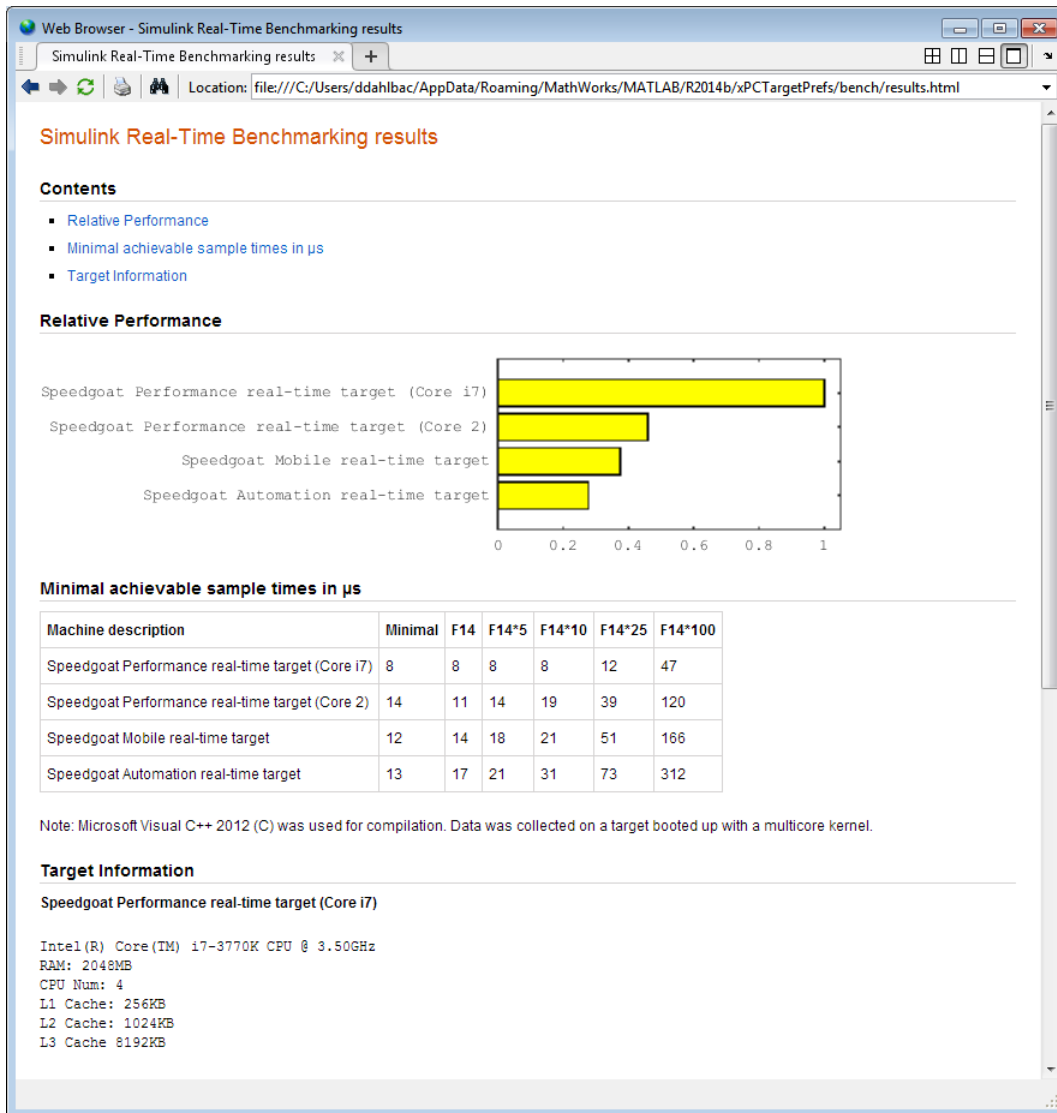
```
slrttest
```

Return an array with representative results for each processor type, in arbitrary order.

```
expected_results = xpcbench();  
expected_results(1)
```

```
ans =
```

```
    Machine: 'Speedgoat Performance real-time target (Core i7)'  
    BenchResults: [1x6 double]  
    Desc: '% Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz  
% RAM: 2...'
```



```
current_results = xpcbench('xpcosc', '-verbose', '-reboot', '-cleanup')
```

Benchmark the target computer using the `xpcosc` model and all control options, and return a structure array with results.

Start the target computer and run confidence test.

`slrttest`

Build 'xpcosc', print build messages, run benchmark, delete build files, restart the target computer, and return results.

```
current_results = xpcbench('xpcosc', '-verbose', '-reboot',
    '-cleanup')

### Starting Simulink Real-Time build procedure for model: xpcosc
### Generating code into build folder: xpcosc_slrt_rtw
### Generated code for 'xpcosc' is up to date because no
    structural, parameter or code replacement library
    changes were found.
.
.
.
### Successful completion of build procedure for model: xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create SimulinkRealTime.target object tg
Target: TargetPC1
    Connected          = Yes
.
.
.
### Running benchmark for model: xpcosc
### Reboot target: TargetPC1..... OK

Benchmark results for model:          xpcosc
Number of blocks in model:            10
Elapsed time for model build (sec):    14.5
Elapsed time for model benchmark (sec): 200.5
Minimal achievable sample time (microsec): 11.9

current_results =
    Name: 'xpcosc'
    nBlocks: 10
    BuildTime: 14.4840
    BenchTime: 200.4516
```

Tsmin: 1.1875e-05

## Input Arguments

**benchmark** — Benchmark name or model name

this | *usermdl* | minimal | f14 | f14\*5 | f14\*10 | f14\*25 | f14\*100

Benchmark, specified as a literal string or string variable containing one of:

this	All five predefined benchmark models (minimal, f14, f14*5, f14*10, f14*25)
<i>usermdl</i>	Your model, <i>usermdl</i> .
minimal	Minimal model consisting of three blocks (Constant, Gain, Termination).
f14	Standard Simulink example f14 (62 blocks, 10 continuous states).
f14*5	Five f14 systems modeled in subsystems (310 blocks, 50 continuous states).
f14*10	Ten f14 systems (620 blocks, 100 continuous states).
f14*25	25 f14 systems (1550 blocks, 250 continuous states).
f14*100	100 f14 systems (6200 blocks, 1000continuous states).

When using function form, enclose literal arguments in single quotes.

Example: 'this'

Example: '-reboot'

Data Types: char

## Output Arguments

**expected\_results** — Results of predefined benchmarks previously run on representative target computers

struct array

Contains representative benchmark results in a structure array with element fields:

<i>Machine</i>	Target computer information string containing CPU type, CPU speed, compiler
<i>BenchResults</i>	Target computer benchmark performance for all five predefined benchmarks
<i>Desc</i>	Target computer descriptor string containing machine type, RAM size, cache size

### **current\_results** – Current results of specified benchmark

struct

Contains actual benchmark results in a structure with fields:

<i>Name</i>	Benchmark name
<i>nBlocks</i>	Number of blocks in benchmark
<i>BuildTime</i>	Elapsed time in seconds to build benchmark
<i>BenchTime</i>	Elapsed time in seconds to run benchmark
<i>Tsmin</i>	Minimal achievable sample time in seconds for benchmark

## More About

### Tips

- Before you run `xpcbench`, you must be able to start the target computer, connect the development computer to the target computer, and run the confidence test, `slrttest`, with no failures.
- After running `xpcbench` on your model and system, set your model sample time to the minimal achievable sample time value reported. Smaller sample times overload the target computer.
- The stored benchmark results were collected with **Multicore CPU support** disabled. When evaluating your system, temporarily disable this target setting using `slrtexplr`.

- The stored benchmark models were compiled using a sampling of the supported compilers. When evaluating your system, find the closest match to the compiler that you are using.
- Benchmark `minimal` has neither continuous nor discrete states. It provides information about the target computer interrupt latencies.
- [http://www.mathworks.com/support/compilers/current\\_release/](http://www.mathworks.com/support/compilers/current_release/)

**See Also**`slrttest`

# xpcbootdisk

Create Simulink Real-Time boot disk or DOS Loader files and confirm current environment properties (not recommended)

## Syntax

xpcbootdisk

## Description

xpcbootdisk creates a Simulink Real-Time boot floppy, CD or DVD boot image, network boot image, or DOS Loader files for the current Simulink Real-Time environment. Use the `setxpcenv` function to set environment properties.

---

**Note:** Command `xpcbootdisk` will be removed in a future release. Use `SimulinkRealTime.createBootImage` instead.

---

What `xpcbootdisk` does depends upon the value of the `TargetBoot` property.

- **BootFloppy** — To create a boot floppy disk, the software prompts you to insert an empty formatted disk into the drive. The software writes the kernel image onto the disk and displays a summary of the creation process.
- **CDBoot** — To create a CD or DVD boot disk, the software prompts you to insert an empty formatted CD or DVD into the drive. The software writes the kernel image onto the CD or DVD and displays a summary of the creation process.
- **NetworkBoot** — To create a network boot image, the software starts the network boot server process.
- **DOSLoader** — To create DOS Loader files, the software writes kernel image and DOS Loader files into a designated location on the development computer. You can then copy the files to the target computer hard drive, to a floppy disk, or to a flash drive.
- **StandAlone** — To create files for a standalone application, you must separately compile and download a combined kernel and real-time application. `SimulinkRealTime.createBootImage` does not generate a standalone application.



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 Simulink Real-Time environment with the function `xpcbootdisk`.

## Examples

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

```
xpcbootdisk
```

## More About

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

## See Also

`getxpcenv` | `setxpcenv` | `xpcnetboot`

## xpcbytes2file

Generate file suitable for use by real-time From File block (not recommended)

### Syntax

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

### Arguments

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

### Description

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

---

**Note:** Command `xpcbytes2file` will be removed in a future release. Use `SimulinkRealTime.utils.bytes2file` instead.

---

If the data is organized such that a row refers to a single time step and not a column, 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 real-time From File block to output a variable `errorval` (single precision, scalar) and `velocity` (double, width 3) at every time step, you can generate the file with the command:

```
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 real-time From File block to output

28 bytes

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

at every sample time.

## xpcexplr

Configure target computer and real-time application for execution (not recommended)

### Syntax

```
xpcexplr
```

### Description

Typing `xpcexplr` at the MATLAB command prompt opens Simulink Real-Time Explorer.

---

**Note:** Command `xpcexplr` will be removed in a future release. Use `slrtexplr` instead.

---

From within Simulink Real-Time Explorer, you can export a session as a standalone executable that runs without MATLAB.

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

- Environment configuration
  - Configure and view communication parameters.
  - Configure target computer settings
  - Configure target computer startup
  - Browse target computer file system.
- Control
  - Load, run, and unload real-time applications on the target computer.
  - Connect to and disconnect from the target computer.
  - Change stop time and sample times without regenerating code.

- Record task execution time during or after last run.
- Instrumentation
  - Create graphical instrument panels for acquiring signals and tuning parameters.
  - Save and load instrument panels.
  - Start and stop instrument panels.
  - Use instrument panels to interact with application.
- Signal acquisition
  - Create, save, and load signal groups.
  - Monitor signals.
  - Add and configure host, target, or file scopes.
  - Attach signals to or remove signals from scopes.
  - Start and stop scopes.
  - Attach signals to instruments.
- Parameter tuning
  - Create, save, and load parameter groups.
  - Display and tune parameters.
  - Attach parameters to instruments.
- Window configuration
  - Make multiple workspaces visible simultaneously.
  - Move workspaces around the window.
  - Export model configuration as a standalone executable.
  - Save and restore model configuration layouts.

When you run Simulink Real-Time Explorer as a standalone executable, it has the following restrictions:

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.
- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. If you rename the target

computer, update the **TargetName** parameter for each instrument to maintain the connection to the real-time application.

- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.
- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.

You cannot move a signal from one signal group to another or create or load a new signal group.

- You can access only parameters in parameter groups that you loaded before you exported the configuration.

You cannot move a parameter from one parameter group to another or create or load a new parameter group.

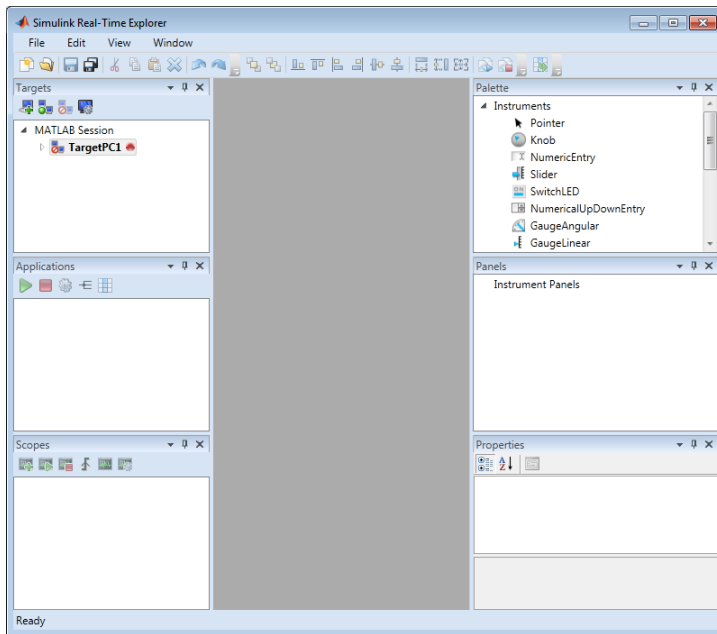
- You cannot save session layouts. If you close a window, you can restore the original layout using **File > Restore Original View**.

## Examples

### Default

Open Simulink Real-Time Explorer

```
xpcexplr
```



- “Ethernet Link Setup”
- “Serial Link Setup”
- “Target Computer Settings”
- “Target Boot Methods”
- “Execute Real-Time Application Using Simulink Real-Time Explorer”
- “Monitor Signals Using Simulink Real-Time Explorer”
- “Create Target Scopes Using Simulink Real-Time Explorer”
- “Create Host Scopes Using Simulink Real-Time Explorer”
- “Create File Scopes Using Simulink Real-Time Explorer”
- “Tune Parameters Using Simulink Real-Time Explorer”
- “Simulink Real-Time Explorer Run Within MATLAB”
- “Simulink Real-Time Explorer Deployed as Standalone Executable”

## More About

- “Instruments and Instrument Panels”

## xpcgetCC

Compiler settings for Simulink Real-Time environment (not recommended)

### 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`.

---

**Note:** Function `xpcgetCC` will be removed in a future release. Use `slrtgetCC` instead.

---

`[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 Simulink Real-Time environment.

`xpcgetCC('installed')` lists the Simulink Real-Time supported compilers installed on the current development computer

`[compilers] = xpcgetCC('installed')` returns the Simulink Real-Time supported compilers installed on the current development computer in a structure.

The `mex -setup` command sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft® compiler. The `slrtgetCC` function



returns the result of the `slrtsetCC` command only, not the result of the `mex` command. If `xpcgetCC` returns an empty string as *location*, Simulink Real-Time uses the MEX compiler.

## Examples

Return the compiler type.

```
type = xpcgetCC
```

Return the compiler type and compiler location.

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

Return the Simulink Real-Time supported compilers installed on the current development 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

Create kernel to boot target computer over dedicated network (not recommended)

### Syntax

```
xpcnetboot  
xpcnetboot targetPCName
```

### Arguments

<i>targetPCName</i>	Target computer name as identified in Simulink Real-Time Explorer.
---------------------	--

### Description

`xpcnetboot` creates a Simulink Real-Time kernel from which a target computer within the same network can start.

---

**Note:** Command `xpcnetboot` will be removed in a future release. Use `SimulinkRealTime.createBootImage` instead.

---

`xpcnetboot` 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` without an argument creates a kernel for the default target computer (as identified in Simulink Real-Time Explorer).

`xpcnetboot targetPCName` creates a Simulink Real-Time kernel and waits for a request from the target computer named *targetPCName* (as identified in Simulink Real-Time Explorer).

## Examples

In the following example, `xpcnetboot` creates a Simulink Real-Time kernel and waits for a request from the target computer, `TargetPC1`.

```
xpcnetboot TargetPC1
```

## xpcsetCC

Compiler settings for Simulink Real-Time environment (not recommended)

### Syntax

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

### Description

`xpcsetCC('setup')` queries the development computer for installed C compilers that the Simulink Real-Time environment supports. You can then select the C compiler.

---

**Note:** Command `xpcsetCC` will be removed in a future release. Use `slrtsetCC` instead.

---

`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.

The command `mex -setup` sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. Use `xpcsetCC -setup` only if you need to specify different compilers for MEX and Simulink Real-Time.

To return to the default compiler from a setting by `xpcsetCC`, type `xpcsetCC('VisualC', '')`, setting the compiler location to the empty string.

### More About

- “Command-Line C Compiler Configuration”

**See Also**  
xpcgetCC

## xpctarget Package

Package for Simulink Real-Time MATLAB classes (not recommended)

### Description

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

---

**Note:** Package `xpctarget` will be removed in a future release. Use package `SimulinkRealTime` methods instead.

---

### Functions

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

Function	Description
<code>xpctarget.fs</code>	Create file system object
<code>xpctarget.ftp</code>	Create file transfer protocol (FTP) object
<code>xpctarget.targets</code>	Create container object to manage target computer environment collection objects
<code>xpctarget.xpc</code>	Create target object representing real-time application

## xpctarget.env Class

Stores target environment properties (not recommended)

### Description

The environment properties define communication between the development and target computers and the type of target boot floppy created during the setup process. An understanding of the environment properties will help you configure the Simulink Real-Time environment.

---

**Note:** Class `xpctarget.env` will be removed in a future release. Use Target Settings Properties instead.

---

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.

## Properties


To read target environment properties from the Command Window, use `xpctarget.targets.Item`. For example:

```
tgs = xpctarget.targets;
env_object = Item(tgs, 'TargetPC1');
property_value = env_object.HostTargetComm
```

To change a property by assignment:

```
tgs = xpctarget.targets;
env_object = Item(tgs, 'TargetPC1');
env_object.HostTargetComm = 'RS232'
```

To access the environment properties in Simulink Real-Time Explorer:

- 1 In the **Targets** pane, expand a target computer node.
- 2 In the toolbar, click the Target Properties icon .
- 3 Expand the sections **Host-to-Target communication**, **Target settings**, or **Boot configuration**.
- .
- .
- .

### Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you must set the other properties that start with TcpIp.</p>



Environment Property	Description
	<b>Note:</b> RS-232 communication type 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, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'.  Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of Simulink Real-Time 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
<p>TcpIpGateway</p>	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of Simulink Real-Time 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.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your development 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.</p>
<p>TcpIpSubNetMask</p>	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>
<p>TcpIpTargetAddress</p>	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>

Environment Property	Description
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of Simulink Real-Time 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 development computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p>

Environment Property	Description
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>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.</p>

Environment Property	Description
TcpIpTargetISAMemPort	<p>Property value is '0xnnnn'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties <b>TcpIpISAMemPort</b> and <b>TcpIpISAIRQ</b>. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>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.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (<b>telnet</b>, <b>ftp</b>, . . .) and is only of use on the target computer.</p>

### Target Settings

Environment Property	Description
EthernetIndex	Property value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the

Environment Property	Description
	<p>(n - 1) th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>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'.</p>
MaxModelSize	<p>Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported.</p> <p>Select 1 MB or 4 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>Setting <b>Model size</b> is enabled for <b>Boot mode Stand Alone</b> only.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the real-time application. Memory not used by the real-time application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.</p>

Environment Property	Description
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
Name	Target computer name.
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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 slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The development computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of Simulink Real-Time Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, real-time application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the real-time application reads the target computer BIOS and determines the amount of memory up to a maximum of 4 GB. If the real-time application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 4 GB.</p> <p>The Simulink Real-Time kernel can use only 4 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>



Environment Property	Description
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <p>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'.</p>

### Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of Simulink Real-Time Explorer.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of Simulink Real-Time Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network.</p> <p>Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p> <p>xx:xx:xx:xx:xx:xx</p>

<b>Environment Property</b>	<b>Description</b>
	<p>To update the MAC address in Simulink Real-Time Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> 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.</p>

## xpctarget.fs Class

Manage the folders and files on the target computer (not recommended)

### Description

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

---

**Note:** Class `xpctarget.fs` will be removed in a future release. Use class `SimulinkRealTime.fileSystem` instead.

---

### Constructor

Constructor	Description
<code>xpctarget.fs</code>	Create file system object

### Methods

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

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

These methods are specific to class `fs`.

Method	Description
<code>xpctarget.fs.diskinfo</code>	Information about target computer drive

<b>Method</b>	<b>Description</b>
<code>xpctarget.fs.fclose</code>	Close open target computer file(s)
<code>xpctarget.fs.fileinfo</code>	Target computer file information
<code>xpctarget.fs.filetable</code>	Information about open files in target computer file system
<code>xpctarget.fs.fopen</code>	Open target computer file for reading
<code>xpctarget.fs.fread</code>	Read open target computer file
<code>xpctarget.fs.fwrite</code>	Write binary data to open target computer file
<code>xpctarget.fs.getfilesize</code>	Size of file on target computer
<code>xpctarget.fs.removefile</code>	Remove file from target computer

# xpctarget.fs

Create Simulink Real-Time file system object (not recommended)

## Syntax

```
filesys_object = xpctarget.fs  
filesys_object = xpctarget.fs(target_object)
```

## Arguments

<code>filesys_object</code>	Variable name to reference the file system object.
<code>target_object</code>	Variable name to reference the target object.

## 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.

---

**Note:** Constructor `xpctarget.fs` will be removed in a future release. Use constructor `SimulinkRealTime.fileSystem` instead.

---

If you have one target computer, or if you designate a target computer as the default one in your system, use `filesys_object = xpctarget.fs` to create a file system object.

If you have a target computer object in the Simulink Real-Time Explorer, use `filesys_object = xpctarget.fs(target_object)` to construct a corresponding file system object from the MATLAB Command Window.

## Examples

In the following example, a file system object for the default target computer is created.

```
fs1 = xpctarget.fs
```

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('TargetPC1');  
fs2 = xpctarget.fs(tg1)
```

# xpctarget.fs.diskinfo

Information about target computer drive (not recommended)

## Syntax

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

## Arguments

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>target_PC_drive</code>	Name of the target computer drive for which to return information.

## Description

`diskinfo(filesys_obj, target_PC_drive)` returns disk information for the specified target computer drive.

This is a method of `xpctarget.fs` objects called from the development computer.

## Examples

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

```
diskinfo(fsys, '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
```



## xpctarget.fs.fclose

Close open target computer files (not recommended)

### Syntax

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

### Arguments

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>file_ID</code>	File identifier of the file to close.

### Description

Method of `xpctarget.fs` objects. From the development 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)
```

### See Also

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

## xpctarget.fs.fileinfo

Target computer file information (not recommended)

### Syntax

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

### Arguments

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>file_ID</code>	File identifier of the file for which to get file information.

### Description

Method of `xpctarget.fs` objects. From the development 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)
ans =
        FilePos: 0
    AllocatedSize: 12288
    ClusterChains: 1
VolumeSerialNumber: 1.0450e+009
        FullName: 'C:\DATA.DAT'
```

## xpctarget.fs.filetable

Information about open files in target computer file system (not recommended)

### Syntax

```
filetable(filesys_obj)
```

### Arguments

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

### Description

Method of `xpctarget.fs` objects. From the development 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.

---

**Note:** Use the `filetable` function only to recover the lost file handle value when MATLAB exits with files still open on the target computer. The function has no other use.

---

### Examples

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

```
filetable(fsys)
ans =
  Index      Handle  Flags      FilePos  Name
-----
      0  00060000  R__         8512  C:\DATA.DAT
      1  00080001  R__           0  C:\DATA1.DAT
      2  000A0002  R__         8512  C:\DATA2.DAT
```

```
3 000C0003 R__      8512 C:\DATA3.DAT
4 001E000S R__      0   C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a 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.

```
fclose(fsys,h1);
```

## See Also

`xpctarget.fs fclose` | `xpctarget.fs fopen`

## xpctarget.fs.fopen

Open target computer file for reading (not recommended)

### Syntax

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

### Arguments

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

### Description

Method of `xpctarget.fs` objects. From the development 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')
ans =
    2883584
d = fread(fsys,h);
```

## See Also

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

# xpctarget.fs.fread

Read open target computer file (not recommended)

## Syntax

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

## Arguments

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

## Description

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

`A = fread(file_obj, 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.

This is a method of `xpctarget.fs` objects called from the development computer.

## Examples

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

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

This reads the file `data.dat` and stores the contents of the file to `d`. This content is in the Simulink Real-Time file format.

## See Also

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



## xpctarget.fs.fwrite

Write binary data to open target computer file (not recommended)

### Syntax

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

### Arguments

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

### Description

Method of `xpctarget.fs` objects. From the development 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

```
fopen(fsys, '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**

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

## xpctarget.fs.getfilesize

Size of file on target computer (not recommended)

### Syntax

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

### Arguments

<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.
<code>file_ID</code>	File identifier of the file to get the size of.

### Description

Method of `xpctarget.fs` objects. From the development computer, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target computer file system. Use the Simulink Real-Time file object method `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)
```

### See Also

`xpctarget.fs.fopen`

## **xpctarget.fs.removefile**

Remove file from target computer (not recommended)

### **Syntax**

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

### **Arguments**

<code>file_name</code>	Name of the file to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

### **Description**

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

You cannot recover this file once it is removed.

---

**Note:** Method `xpctarget.fs.removefile` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.removefile` instead.

---

### **Examples**

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

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

## xpctarget.fs.selectdrive

Select target computer drive (not recommended)

### Syntax

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

### Arguments

<code>drive</code>	Name of the drive to set.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> 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:** Method `xpctarget.fs.selectdrive` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.selectdrive` or `SimulinkRealTime.fileSystem.cd` instead.

---

### Examples

Set the current target computer drive to `D:\`.

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

## xpctarget.fsbase Class

Base class of file system and file transfer protocol (FTP) classes (not recommended)

### 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:

---

**Note:** Class `xpctarget.fsbase` will be removed in a future release. Use class `SimulinkRealTime.fileSystem` instead.

---

### Methods

These methods are inherited by the derived classes.

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

# xpctarget.fsbase.cd

Change folder on target computer (not recommended)

## Syntax

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

## Arguments

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<code>target_PC_dir</code>	Name of the target computer folder to change to.

## Description

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

---

**Note:** Method `xpctarget.fsbase.cd` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.cd` or `SimulinkRealTime.fileSystem.selectdrive` instead.

---

## Examples

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

```
cd(fsys, logs)
```

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

```
cd(f, logs)
```

## See Also

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

## **xpctarget.fsbasedir**

List contents of current folder on target computer (not recommended)

### **Syntax**

```
dir(file_obj)
```

### **Arguments**

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

### **Description**

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

---

**Note:** Method `xpctarget.fsbasedir` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.dir` instead.

---

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)
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

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

## **xpctarget.fsbase.mkdir**

Make folder on target computer (not recommended)

### **Syntax**

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

### **Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<code>dir_name</code>	Name of the folder to be created.

### **Description**

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

---

**Note:** Method `xpctarget.fsbase.mkdir` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.mkdir` instead.

---

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

### **Examples**

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

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

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

mkdir(f,logs)

**See Also**

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

## xpctarget.fsbase.pwd

Current folder path of target computer (not recommended)

### Syntax

```
pwd(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. Returns the pathname of the current target computer folder.

---

**Note:** Method `xpctarget.fsbase.cd` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.pwd` instead.

---

### Examples

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

```
pwd(fsys)
```

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

```
pwd(f)
```

### See Also

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

## xpctarget.fsbase.rmdir

Remove folder from target computer (not recommended)

### Syntax

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

### Arguments

<code>dir_name</code>	Name of the folder to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

### Description

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

You cannot recover this folder once it is removed.

---

**Note:** Method `xpctarget.fsbase.rmdir` will be removed in a future release. Use method `SimulinkRealTime.fileSystem.rmdir` instead.

---

### Examples

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

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

## xpctarget.ftp Class

Manage the folders and files on the target computer via file transfer protocol (FTP) (not recommended)

### Description

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

---

**Note:** Class `xpctarget.ftp` will be removed in a future release. Use class `SimulinkRealTime.fileSystem` instead.

---

### Constructor

Constructor	Description
<code>xpctarget.ftp</code>	Create file transfer protocol (FTP) object

### Methods

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

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

These methods are specific to class `ftp`.

Method	Description
<code>xpctarget.ftp.get</code>	Retrieve copy of requested file from target computer

<b>Method</b>	<b>Description</b>
xpctarget.ftp.put	Copy file from development computer to target computer

## xpctarget.ftp

Create file object (not recommended)

### Syntax

```
file_object = xpctarget.ftp  
file_object = xpctarget.ftp(target_object)
```

### Arguments

file_object	Variable name to reference the file object.
target_object	Variable name to reference the target object.

### Description

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

---

**Note:** Constructor `xpctarget.ftp` will be removed in a future release. Use constructor `SimulinkRealTime.fileSystem` instead.

---

If you have one target computer, or if you designate a target computer as the default one in your system, use `file_object = xpctarget.ftp` to create a file object.

If you have a target computer object in the Simulink Real-Time Explorer, use `file_object = xpctarget.ftp(target_object)` to construct a corresponding file object from the MATLAB Command Window.

### Examples

In the following example, a file object for the default target computer is created.



```
ftp1=xpctarget.ftp
```

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

```
tg1=xpctarget.xpc('TargetPC1');  
ftp2=xpctarget.ftp(tg1)
```

## xpctarget.ftp.get

Retrieve copy of requested file from target computer (not recommended)

### Syntax

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

### Arguments

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	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 development computer. `file_name` must be either a fully qualified filename on the target computer, or located in the current folder of the target computer.

---

**Note:** Method `xpctarget.ftp.get` will be removed in a future release. Use method `SimulinkRealTime.copyFileToHost` instead.

---

### 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')  
ans = data.dat
```

### See Also

`xpctarget.ftp.put`

## xpctarget.ftp.put

Copy file from development computer to target computer (not recommended)

### Syntax

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

### Arguments

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	Name of the file to copy to the target computer.

### Description

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

---

**Note:** Method `xpctarget.ftp.put` will be removed in a future release. Use method `SimulinkRealTime.copyFileToTarget` instead.

---

`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 development computer to the current folder of the target computer FTP object `f`.

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

### See Also

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

## xpctarget.targets Class

Container object to manage target computer environment collection objects (not recommended)

### Description

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

---

**Note:** Class `xpctarget.targets` will be removed in a future release. Use package `SimulinkRealTime` methods instead.

---

### Constructor

Constructor	Description
<code>xpctarget.targets</code>	Create container object to manage target computer environment collection objects

### Methods

Method	Description
<code>xpctarget.targets.Add (env collection object)</code>	Add a new Simulink Real-Time environment collection object.
<code>xpctarget.targets.getTargetName (env collection object)</code>	Retrieve the Simulink Real-Time environment collection object names.
<code>xpctarget.targets.Item (env collection object)</code>	Retrieve Simulink Real-Time environment collection object.
<code>xpctarget.targets.makeDefault (env collection object)</code>	Set target computer environment collection object as default.
<code>xpctarget.targets.Remove (env collection object)</code>	Remove a Simulink Real-Time environment collection object.

## Properties

To get the value of a readable property from the targets object:

```
value = targets_object.property_name
```

For example, to get the NumTargets property of the targets object:

```
targets = xpctarget.targets;
value = targets.NumTargets
```

To set the value of a writable property from a targets object:

```
targets_object.property_name = new_value
```

For example, to set the ShowTargets of the targets object:

```
targets = xpctarget.targets;
targets.ShowTargets = 'on'
```

Property	Description	Writable
DefaultTarget	Returns an <code>xpctarget.env</code> object that references the default target computer object environment.	No
NumTargets	Returns the number of target computer environment objects in the container.	No
ShowTargets	When <code>on</code> (the default) shows information about the targets. When <code>off</code> , suppresses information.	Yes

## xpctarget.targets

Create container object to manage target computer environment collection objects (not recommended)

### Syntax

```
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 Simulink Real-Time system.

---

**Note:** Constructor `xpctarget.targets` will be removed in a future release.

---

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 the `env_collection_object.propertyname` syntax.

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 the targets in your system.

tgs=xpctarget.targets

## **xpctarget.targets.Add (env collection object)**

Add new Simulink Real-Time environment collection object (not recommended)

### **Syntax**

```
env_collection_object.Add
```

### **Description**

Method of `xpctarget.targets` objects. `Add` creates a Simulink Real-Time environment collection object on the development computer.

---

**Note:** Method `xpctarget.targets.Add (env collection object)` will be removed in a future release. Use method `SimulinkRealTime.addTarget` instead.

---

### **Examples**

Add a new Simulink Real-Time 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);  
Add(tgs);  
get(tgs);
```

### **See Also**

`xpctarget.targets`



## xpctarget.targets.getTargetNames (env collection object)

Retrieve Simulink Real-Time environment object names (not recommended)

### Syntax

```
env_collection_object.getTargetNames
```

### Description

Method of `xpctarget.targets` objects. `getTargetNames` retrieves the names of the existing Simulink Real-Time environment collection objects from the `xpctarget.targets` class.

---

**Note:** Method `xpctarget.targets.getTargetNames (env collection object)` will be removed in a future release. Use package `SimulinkRealTime` methods instead.

---

### Examples

Retrieve the names of the Simulink Real-Time environment collection objects in the system. Assume that `tgs` represents the target object collection environment.

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

### See Also

`xpctarget.targets`

## **xpctarget.targets.Item (env collection object)**

Retrieve specific Simulink Real-Time environment (env) object (not recommended)

### **Syntax**

```
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.

---

**Note:** `xpctarget.targets.Item (env collection object)` will be removed in a future release. Use `SimulinkRealTime.getTargetSettings` instead.

---

### **Examples**

Retrieve a new Simulink Real-Time environment collection object from the system. Assume that `tgs` represents the target object collection environment.

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

### **See Also**

`xpctarget.targets`

## xpctarget.targets.makeDefault (env collection object)

Set specific target computer environment object as default (not recommended)

### Syntax

```
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.

---

**Note:** `xpctarget.targets.makeDefault (env collection object)` will be removed in a future release. Use `SimulinkRealTime.targetSettings.setAsDefaultTarget` instead.

---

### 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);  
getTargetNames(tgs)  
makeDefault(tgs, 'TargetPC2')
```

### See Also

`xpctarget.targets`

## **xpctarget.targets.Remove (env collection object)**

Remove specific Simulink Real-Time environment object (not recommended)

### **Syntax**

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

### **Description**

Method of `xpctarget.targets` objects. `Remove` removes an existing Simulink Real-Time 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.

---

**Note:** `xpctarget.targets.Remove (env collection object)` will be removed in a future release. Use `SimulinkRealTime.removeTarget` instead.

---

### **Examples**

Remove a Simulink Real-Time environment collection object from the system. Assume that `tgs` represents the target object collection environment.

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

### **See Also**

`xpctarget.targets`

## xpctarget.xpc Class

Target object representing real-time application (not recommended)

### Description

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

---

**Note:** Class `xpctarget.xpc` will be removed in a future release. Use class `SimulinkRealTime.target` instead.

---

### Constructor

Constructor	Description
<code>xpctarget.xpc</code>	Create target object representing real-time application

### Methods

Method	Description
<code>xpctarget.xpc.addscope</code>	Create scopes
<code>xpctarget.xpc.close</code>	Close serial port connecting development computer with target computer
<code>xpctarget.xpc.getlog</code>	All or part of output logs from target object
<code>xpctarget.xpc.getparam</code>	Value of target object parameter index
<code>xpctarget.xpc.getparaml</code>	Parameter index from parameter list
<code>xpctarget.xpc.getparamr</code>	Block path and parameter name from index list
<code>xpctarget.xpc.getscope</code>	Scope object pointing to scope defined in kernel
<code>xpctarget.xpc.getsignal</code>	Value of target object signal index
<code>xpctarget.xpc.getsignalr</code>	Signal index or signal property from signal list

Method	Description
<code>xpctarget.xpc.getsignals</code>	Return vector of signal indices
<code>xpctarget.xpc.getsignal</code>	Return signal label
<code>xpctarget.xpc.getsignal</code>	Signal name from index list
<code>xpctarget.xpc.load</code>	Download real-time application to target computer
<code>xpctarget.xpc.loadparameter</code>	Restore parameter values saved in specified file
<code>xpctarget.xpc.reboot</code>	Reboot target computer
<code>xpctarget.xpc.remscope</code>	Remove scope from target computer
<code>xpctarget.xpc.saveparameter</code>	Save current real-time application parameter values
<code>xpctarget.xpc.setparameter</code>	Change writable target object parameters
<code>xpctarget.xpc.start</code> (real-time application object)	Start execution of real-time application on target computer
<code>xpctarget.xpc.stop</code> (real-time application object)	Stop execution of real-time application on target computer
<code>xpctarget.xpc.targetping</code>	Test communication between development and target computers
<code>xpctarget.xpc.unload</code>	Remove current real-time application from target computer

## Properties

To get the value of a readable target object property from a target object:

```
value = target_object.property_name
```

For example, to get the `CommunicationTimeOut` of the target object:

```
target_object = xpctarget.xpc;
value = target_object.CommunicationTimeOut
```

To set the value of a writable target object property from a target object:

```
target_object.property_name = new_value
```

For example, to set the `CommunicationTimeOut` of the target object:

```
target_object = xpctarget.xpc;
```

```
target_object.CommunicationTimeOut = 10
```

At the target computer command line, you can set the target object properties `stoptime`, `sampletime`, and writable model parameters.

```
stoptime = floating_point_number
sampletime = floating_point_number
setpar parameter_index = parameter_value
```

Property	Description	Writable
Application	Name of the Simulink model and real-time application built from that model.	No
AvgTET	<p>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.</p> <p>The TET includes:</p> <ul style="list-style-type: none"> <li>• Complete I/O latency.</li> <li>• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.</li> <li>• Asynchronous interruptions.</li> <li>• Parameter updating latency (if the <b>Double buffer parameter changes</b> parameter is set in the <b>Simulink Real-Time Options</b> node of the model Configuration Parameters dialog box).</li> </ul> <p>Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:</p> <ul style="list-style-type: none"> <li>• Time required to measure TET</li> </ul>	No

Property	Description	Writable
	<ul style="list-style-type: none"> <li>Interrupt latency required to schedule and run one step of the model</li> </ul>	
CommunicationTimeout	Communication timeout between the development and target computers, in seconds.	Yes
Connected	Communication status between the development computer and the target computer. Values are 'Yes' and 'No'.	No
CPUOverload	CPU status for overload. If the real-time 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 real-time application started running. When the real-time application stops, the total execution time is displayed.	No
LogMode	Controls which data points are logged: <ul style="list-style-type: none"> <li>Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li> <li>Value-equidistant logging. Logs a data point only when an output signal from the <b>OutputLog</b> changes by a specified value (increment). Set the value to the difference in signal values.</li> </ul>	Yes



Property	Description	Writable
MaxLogSamples	<p>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.</p> <p>This value is calculated by dividing the <b>Signal Logging Buffer Size</b> by the number of logged signals. The <b>Signal Logging Buffer Size</b> box is in the <b>Simulink Real-Time Options</b> pane of the Configuration Parameters dialog box.</p>	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	<p>Type of Simulink Coder™ code generation. Values are 'Real-Time Singletasking' and 'Real-Time Multitasking'. The default value is 'Real-Time Singletasking'.</p> <p>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.</p>	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

Property	Description	Writable
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 real-time application.	No
Parameters	List of tunable parameters. This list is visible only when ShowParameters is set to 'on': <ul style="list-style-type: none"> <li>• Property value. Value of the parameter in a Simulink block.</li> <li>• Type. Data type of the parameter. Always double.</li> <li>• Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix.</li> <li>• Parameter name. Name of a parameter in a Simulink block.</li> <li>• Block name. Name of a Simulink block.</li> </ul>	No
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “Alternative Configuration and Control Methods” for limitations on target property changes to sample times.)	Yes
Scopes	List of index numbers, with one index for each scope.	No
SessionTime	Time since the kernel started running on your target computer. This is also the elapsed time since you 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'. <ul style="list-style-type: none"> <li>Property name. S0, S1. . .</li> <li>Property value. Value of the signal.</li> <li>Block name. Name of the Simulink block the signal is from.</li> </ul>	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the real-time application.	No
Status	Execution status of your real-time application. Values are 'stopped' and 'running'.	No
StopTime	Time when the real-time application stops running. Values are in seconds. The original value is set in the <b>Solver</b> 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 real-time application.  To enable logging of the TET, you need to select the <b>Log Task Execution Time</b> check box in the <b>Simulink Real-Time Options</b> 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 real-time application.	No

<b>Property</b>	<b>Description</b>	<b>Writable</b>
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 <b>enabled</b> .	Yes

## xpctarget.xpc

Create target object representing real-time application (not recommended)

### Syntax

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

### Arguments

target_object	Variable name to reference the target object
target_name	Target name as specified in the Simulink Real-Time Explorer

### Description

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

---

**Note:** Constructor `xpctarget.xpc` will be removed in a future release. Use constructor `SimulinkRealTime.target` or function `slrt` instead.

---

If you have one target computer, or if you designate a target computer as the default one in your system, use `target_object=xpctarget.xpc`.

If you have a target computer object in the Simulink Real-Time Explorer, use `target_object=xpctarget.xpc('target_name')` to construct a corresponding target object from the MATLAB Command Window.

## Examples

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

```
tg = xpctarget.xpc
Target: TargetPC1
    Connected          = Yes
    Application        = loader
```

```
tg.targetping
```

```
ans =
```

```
success
```

If you have a Simulink Real-Time 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.targetping`

# xpctarget.xpc.addscope

Create scopes (not recommended)

## Syntax

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

---

**Note:** Method `xpctarget.xpc.addscope` will be removed in a future release. Use method `SimulinkRealTime.target.addscope` instead.

---

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

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

```
scope_object = addscope(target_object,  
                        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
```

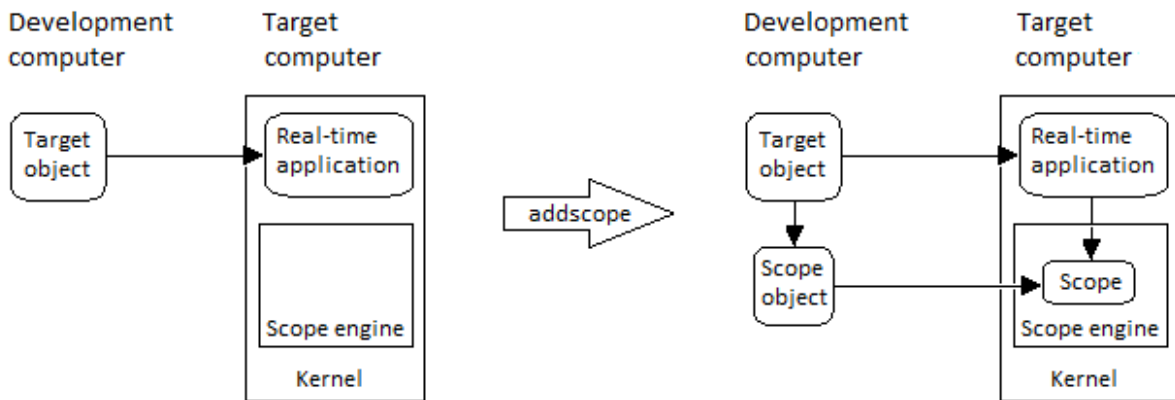
## Arguments

<code>target_object</code>	Name of a target object. The default target name is <code>tg</code> .
<code>scope_type</code>	Values are <code>'host'</code> , <code>'target'</code> , or <code>'file'</code> . This argument is optional with <code>host</code> as the default value.
<code>scope_number</code>	Vector of new scope indices. This argument is optional. The next available integer in the target object property <code>Scopes</code> as the default value.

If you enter a scope index for an existing scope object, the result is an error.

## Description

`addscope` creates a scope of the specified type and updates the target object property `Scopes`. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The Simulink Real-Time product supports 10 target scopes, 8 file scopes, and as many host scopes as the target computer resources can support. If you try to add a scope with the same index as an existing scope, the result is an error.



## Examples

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

```
sc1 = addscope(tg, '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 development 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)
sc1 = getscope(tg, 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 development 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 development 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)
```

## More About

- “Target Scope Usage”
- “Host Scope Usage”
- “File Scope Usage”
- “Application and Driver Scripts”

## See Also

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

## **xpctarget.xpc.close**

Close serial port connecting development computer with target computer (not recommended)

### **Syntax**

```
close(target_object)
```

### **Arguments**

`target_object`                      Name of a target object.

### **Description**

`close` closes the serial link between the development and target computers. 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.

---

**Note:** Method `xpctarget.xpc.close` will be removed in a future release. Use method `SimulinkRealTime.target.close` instead.

---

## xpctarget.xpc.getlog

All or part of output logs from target object (not recommended)

### Syntax

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

### Arguments

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

---

**Note:** Method `xpctarget.xpc.getlog` will be removed in a future release. Use method `SimulinkRealTime.target.getlog` instead.

---

### 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)
```

## More About

- “Set Configuration Parameters”

## xpctarget.xpc.getparam

Value of target object parameter index (not recommended)

### Syntax

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

### Arguments

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

### Description

getparam returns the value of the parameter associated with parameter\_index.

---

**Note:** Method xpctarget.xpc.getparam will be removed in a future release. Use method SimulinkRealTime.target.getparam instead.

---

### Examples

Get the value of parameter index 5.

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

## xpctarget.xpc.getparamid

Parameter index from parameter list (not recommended)

### Syntax

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

### Arguments

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

### Description

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

---

**Note:** Method xpctarget.xpc.getparamid will be removed in a future release. Use method SimulinkRealTime.target.getparamid instead.

---

### 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.

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

Get the property index of a single block.

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

## More About

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

## See Also

`xpctarget.xpc.getsignalid`

## xpctarget.xpc.getparamname

Block path and parameter name from index list (not recommended)

### Syntax

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

### Arguments

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

### Description

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

---

**Note:** Method `xpctarget.xpc.getparamid` will be removed in a future release. Use method `SimulinkRealTime.target.getparamid` instead.

---

### Examples

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

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



## xpctarget.xpc.getscope

Scope object pointing to scope defined in kernel (not recommended)

### Syntax

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

### Arguments

target_object	Name of a target object.
scope_number_vector	Vector of existing scope indices listed in the target object property <code>Scopes</code> . 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.

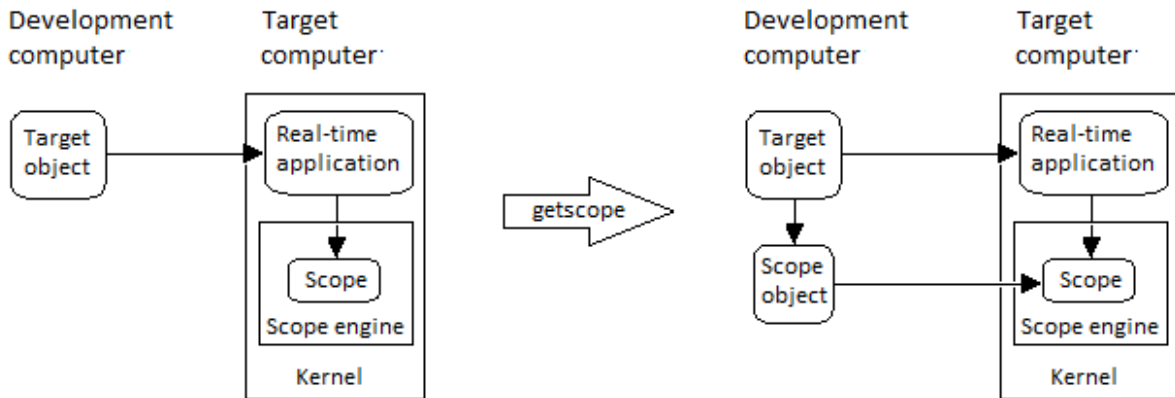
### Description

`getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can find the number of existing scopes using the property `target_object.Scopes`.

---

**Note:** Method `xpctarget.xpc.getscope` will be removed in a future release. Use method `SimulinkRealTime.target.getscope` instead.

---



## Examples

If your Simulink model has a Simulink Real-Time scope block, a target scope is created at the time the real-time 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)
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 development computer. If the target object has more than one scope, it create a vector of scope objects.

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

## More About

- “Application and Driver Scripts”

## See Also

`getxpcenv` | `xpctarget.xpc.remscope`

## xpctarget.xpc.getsignal

Value of target object signal index (not recommended)

### Syntax

```
getsignal(target_object, signal_index)
```

### Arguments

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>signal_index</code>	Index number of the signal.

### Description

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

---

**Note:** Method `xpctarget.xpc.getsignal` will be removed in a future release. Use method `SimulinkRealTime.target.getsignal` instead.

---

### Examples

Get the value of signal index 2.

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

## xpctarget.xpc.getsignalid

Signal index or signal property from signal list (not recommended)

### Syntax

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

### Arguments

<code>target_object</code>	Name of an existing target object.
<code>signal_name</code>	Enter the name of a signal from your Simulink model. For blocks with a single signal, the <code>signal_name</code> is equal to the <code>block_name</code> . For blocks with multiple signals, the Simulink Real-Time software appends S1, S2 . . . to the <code>block_name</code> .

### 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.

---

**Note:** Method `xpctarget.xpc.getsignalid` will be removed in a future release. Use method `SimulinkRealTime.target.getsignalid` instead.

---

### Examples

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

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

## More About

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

## See Also

xpctarget.xpc.getparamid

## xpctarget.xpc.getsignalidsfromlabel

Return vector of signal indices (not recommended)

### Syntax

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

### Arguments

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

### Description

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

---

**Note:** Method `xpctarget.xpc.getsignalidsfromlabel` will be removed in a future release. Use method `SimulinkRealTime.target.getsignalidsfromlabel` instead.

---

### 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

Return signal label (not recommended)

## Syntax

```
getsignallabel(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

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

---

**Note:** Method `xpctarget.xpc.getsignallabel` will be removed in a future release. Use method `SimulinkRealTime.target.getsignallabel` instead.

---

## Examples

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

## See Also

`xpctarget.xpc.getsignalidsfromlabel`

## xpctarget.xpc.getsignalname

Signal name from index list (not recommended)

### Syntax

```
getsignalname(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

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

---

**Note:** Method `xpctarget.xpc.getsignalname` will be removed in a future release. Use method `SimulinkRealTime.target.getsignalname` instead.

---

### Examples

Get the signal name of signal ID 2.

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



# load

Download real-time application to target computer (not recommended)

## Syntax

```
target_object = load(target_object, target_application)
```

## Description

`target_object = load(target_object, target_application)` loads the application `target_application` onto the target computer represented by `target_object`.

---

**Note:** Method `xpctarget.xpc.load` will be removed in a future release. Use method `SimulinkRealTime.target.load` instead.

---

The call returns `target_object`, updated with the new state of the target.

## Input Arguments

### **target\_object** — Object representing target computer

Object of type `xpctarget.xpc` that represents the target computer. Before calling this function, make sure that you have started the target computer with the Simulink Real-Time kernel and have applied the required Ethernet link settings.

Data Types: `struct`

### **real\_time\_application** — Name of real-time application

Name of the real-time application, without file extension. `target_application` can also contain the absolute path to the real-time application, without file extension.

You must build the application in the current working folder on the development computer. By default, the Simulink Real-Time software calls `xpctarget.xpc.load`

automatically after the Simulink Coder build process completes. If a real-time application was previously loaded, before downloading the new real-time application, `xpctarget.xpc.load` unloads the old real-time application.

If you are running the real-time application in Standalone mode, a call to `xpctarget.xpc.load` has no effect. To load a new application, you must rebuild the standalone application files with the new application and transfer the updated files to the target computer using `xpctarget.ftp`. Then, restart the target computer with the new standalone application.

Data Types: char

## Examples

### Load `xpcosc`

Load the real-time application `xpcosc` into target computer `TargetPC1`, represented by target object `tg`. Start the application.

Get the target object.

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

```
Target: TargetPC1
  Connected           = Yes
  Application         = loader
```

Load the real-time application.

```
load(tg, 'xpcosc')
```

```
Target: TargetPC1
  Connected           = Yes
  Application         = xpcosc
  Mode                = Real-Time Single-Tasking
  Status              = stopped
  CPUOverload         = none

  ExecTime            = 0.0000
  SessionTime         = 918.5713
  StopTime            = 0.200000
  SampleTime          = 0.000250
  AvgTET              = NaN
```

```
MinTET           = 9999999.000000
MaxTET           = 0.000000
ViewMode         = 0

TimeLog          = Vector(0)
StateLog         = Matrix (0 x 2)
OutputLog        = Matrix (0 x 2)
TETLog           = Vector(0)
MaxLogSamples    = 16666
NumLogWraps      = 0
LogMode          = Normal

Scopes           = No Scopes defined
NumSignals       = 7
ShowSignals      = off

NumParameters    = 7
ShowParameters   = off
```

Start the application.

```
start(tg)
```

- “Application and Driver Scripts”

## See Also

xpctarget.xpc.unload

## **xpctarget.xpc.loadparamset**

Restore parameter values saved in specified file (not recommended)

### **Syntax**

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

### **Arguments**

<code>target_object</code>	Name of an existing target object.
<code>filename</code>	Enter the name of the file that contains the saved parameters.

### **Description**

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

---

**Note:** Method `xpctarget.xpc.loadparamset` will be removed in a future release. Use method `SimulinkRealTime.target.loadparamset` instead.

---

### **See Also**

`xpctarget.xpc.saveparamset`

# xpctarget.xpc.reboot

Reboot target computer (not recommended)

## 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 Simulink Real-Time kernel is reloaded.

---

**Note:** Method `xpctarget.xpc.reboot` will be removed in a future release. Use method `SimulinkRealTime.target.reboot` instead.

---

On the target computer command line, you can use the corresponding command `reboot`.

You can also use this method to reboot the target computer back to Windows<sup>®</sup> 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

Remove scope from target computer (not recommended)

### Syntax

#### MATLAB command line

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

#### Target computer command line

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

### Arguments

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

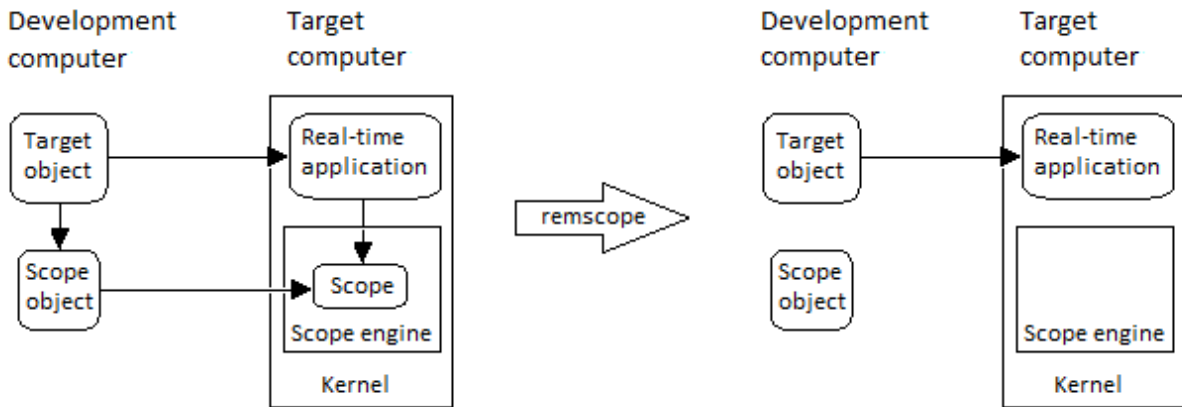
### Description

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 development computer is not deleted.

---

**Note:** Method `xpctarget.xpc.remscope` will be removed in a future release. Use method `SimulinkRealTime.target.remscope` instead.

---



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)
```

Remove two scopes.

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

Remove all scopes.

```
remscope(tg)
```

## More About

- “Application and Driver Scripts”

## See Also

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



## xpctarget.xpc.saveparamset

Save current real-time application parameter values (not recommended)

### Syntax

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

### Arguments

<code>target_object</code>	Name of an existing target object.
<code>filename</code>	Enter the name of the file to contain the saved parameters.

### Description

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

---

**Note:** Method `xpctarget.xpc.saveparamset` will be removed in a future release. Use method `SimulinkRealTime.target.saveparamset` instead.

---

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

### See Also

`xpctarget.xpc.loadparamset`

## xpctarget.xpc.setparam

Change writable target object parameters (not recommended)

### Syntax

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

### Arguments

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

### Description

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

- parIndexVec
- OldValues
- NewValues

---

**Note:** Method `xpctarget.xpc.setparam` will be removed in a future release. Use method `SimulinkRealTime.target.setparam` instead.

---

### 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]}
```

## **xpctarget.xpc.start (real-time application object)**

Start execution of real-time application on target computer (not recommended)

### **Syntax**

**MATLAB command line**

```
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 real-time application represented by the target object. Before using this method, the real-time application must be created and loaded on the target computer. If a real-time application is running, this command has no effect.

---

**Note:** Method `xpctarget.xpc.start (real-time application object)` will be removed in a future release. Use method `SimulinkRealTime.target.start` instead.

---

### **Examples**

Start the real-time application represented by the target object `tg`.

```
start(tg)
```

## **See Also**

`xpctarget.xpc.stop` (real-time application object) | `load` (`xpctarget.xpc`) |  
`xpctarget.xpc.unload` | `xpctarget.xpcsc.stop` (scope object)

## **xpctarget.xpc.stop (real-time application object)**

Stop execution of real-time application on target computer (not recommended)

### **Syntax**

**MATLAB command line**

```
stop(target_object)
```

**Target computer command line**

```
stop
```

### **Arguments**

`target_object`            Name of a target object.

### **Description**

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

---

**Note:** Method `xpctarget.xpc.stop (real-time application object)` will be removed in a future release. Use method `SimulinkRealTime.target.stop` instead.

---

### **Examples**

Stop the real-time application represented by the target object `tg`.

```
stop(tg)
```

### **See Also**

```
xpctarget.xpc.start (real-time application object) |  
xpctarget.xpcsc.start (scope object) | xpctarget.xpcsc.stop (scope  
object)
```

## xpctarget.xpc.targetping

Test communication between development and target computers (not recommended)

### Syntax

```
targetping(target_object)
```

### Arguments

`target_object`     Name of a target object.

### Description

Method of a target object. Use this method to ping a target computer from the development computer. This method returns **success** if the Simulink Real-Time kernel is loaded and running and communication is working between the development and target computers, otherwise it returns **failed**.

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

---

#### Note:

- Method `xpctarget.xpc.targetping` will be removed in a future release. Use command `slrtpingtargetor` method `SimulinkRealTime.target.ping` instead.
  - RS-232 communication type 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)
```

**See Also**

`xpctarget.xpc`



## xpctarget.xpc.unload

Remove current real-time application from target computer (not recommended)

### Syntax

```
unload(target_object)
```

### Arguments

`target_object`      Name of a target object that represents a real-time application.

### Description

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

---

**Note:** Method `xpctarget.xpc.unload` will be removed in a future release. Use method `SimulinkRealTime.target.unload` instead.

---

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 real-time application represented by the target object `tg`.

```
unload(tg)
```

### See Also

`load(xpctarget.xpc)` | `xpctarget.xpc.reboot`

## xpctarget.xpcfs Class

Control and access properties of file scopes (not recommended)

### 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.

---

**Note:** Class `xpctarget.xpcfs` will be removed in a future release. Use class `SimulinkRealTime.fileScope` instead.

---

### Methods

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

Method	Description
<code>xpctarget.xpcsc.addsigr</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.remsigr</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	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

Property	Description	Writable
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <code>TriggerMode</code> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>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.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
TriggerLevel	If <code>TriggerMode</code> 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	<p>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.</p> <p>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.</p>	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
Type	<p>Determines whether the scope is displayed on the development computer or on the target computer. Values are 'Host', 'Target', and 'File'.</p> <p>Property Type is set only once, when the scope is created on the target computer.</p>	No

These properties are specific to class `xpcfs`.

Property	Description	Writeable
AutoRestart	<p>Values are 'on' and 'off'.</p> <p>For file scopes, enable the file scope to collect data up to the number of samples (<code>NumSamples</code>), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b>, then stop.</p> <p>If the named signal data file already exists when you start the real-time application, the software overwrites the old data with the new signal data.</p> <p>To use the <code>DynamicFileName</code> property, set <code>AutoRestart</code> to 'on' first.</p> <p>For host or target scopes, this parameter has no effect.</p>	No
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use <code>DynamicFileName</code>, set <code>AutoRestart</code> to 'on' first. When you enable <code>DynamicFileName</code>, configure <code>Filename</code> 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.</p>	Yes

<b>Property</b>	<b>Description</b>	<b>Writeable</b>
	<p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or target scopes, this parameter has no effect.</p>	

Property	Description	Writeable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named <code>C:\data.dat</code> for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to <code>FileName</code>. 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, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>If you set <code>DynamicFileName</code> and <code>AutoRestart</code> to 'on', configure <code>Filename</code> to dynamically increment. Use a base file name, an underscore (<code>_</code>), and a <code>&lt; &gt;</code> specifier. Within the specifier, enter one to eight <code>%</code> symbols. Each symbol <code>%</code> represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for <code>Filename</code>, <code>C:\work\file_&lt;%%&gt;.dat</code> creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre> <p>The last file name of this series will be <code>file_999.dat</code>. 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</p>	No

Property	Description	Writeable
	<p>data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
MaxWriteFileS	<p>Provide the maximum size of <code>Filename</code>, in bytes. This value must be a multiple of <code>WriteSize</code>. Default is 536870912.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
Mode	<p><b>Note:</b> The <code>Mode</code> property will be removed in a future release.</p> <ul style="list-style-type: none"> <li>• For target scopes, use <code>DisplayMode</code>.</li> <li>• For file scopes, use <code>WriteMode</code>.</li> <li>• For host scopes, this parameter has no effect.</li> </ul>	Yes



Property	Description	Writeable
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
WriteSize	<p>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 better performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

## xpctarget.xpcsc.addsignal

Add signals to scope represented by scope object (not recommended)

### Syntax

#### MATLAB command line

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

#### Target command line

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

### Arguments

<code>scope_object_vector</code>	Name of a single scope object or the name of a vector of scope objects.
<code>signal_index_vector</code>	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
<code>scope_index</code>	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.
- Method `xpctarget.xpcsc.addsignal` will be removed in a future release. Use methods `SimulinkRealTime.targetScope.addsignal`,

`SimulinkRealTime.hostScope.addsignal`, and  
`SimulinkRealTime.fileScope.addsignal` instead.

---

## 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])
```

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
```

## More About

- “Target Scope Usage”
- “Host Scope Usage”
- “File Scope Usage”
- “Application and Driver Scripts”

## See Also

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

## xpctarget.xpcsc Class

Base class for the scope classes (not recommended)

### Description

This is the base class for the scope classes, `xpctarget.xpcfs Class`, `xpctarget.xpcschost Class`, and `xpctarget.xpcscctg 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. The scope class constructors are `Private` and are not intended to be called from the MATLAB prompt.

---

**Note:** Class `xpctarget.xpcsc` will be removed in a future release. Use classes `SimulinkRealTime.targetScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.fileScope` instead.

---

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

### Methods

These methods are inherited by the derived classes.

Method	Description
<code>xpctarget.xpcsc.addsig</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.remsigr</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software trigger start of data acquisition for scope(s)

## Properties

Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the real-time application.

To get the value of a readable scope object property from a scope object:

```
scope_object = getscope(target_object, scope_number);
value = scope_object.scope_object_property
```

For example, to get the **Decimation** of scope 3:

```
scope_object = getscope(tg, 3);
value = scope_object.Decimation
```

To set the value of a writable scope property from a scope object:

```
scope_object = getscope(target_object, scope_number);
scope_object.scope_object_property = new_value
```

For example, to set the **Decimation** of scope 3:

```
scope_object = getscope(tg, 3);
scope_object.Decimation = 10
```

Not all properties are user-writable. For example, property **Type** is not writable after you have created the scope.

The properties in the following table apply to file, host, and target scopes.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes

Property	Description	Writable
NumSamples	<p>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.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
TriggerLevel	If <b>TriggerMode</b> is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes

Property	Description	Writable
TriggerSample	<p>If <code>TriggerMode</code> is 'Scope', then <code>TriggerSample</code> specifies which sample of the triggering scope the current scope should trigger on. For example, if <code>TriggerSample</code> 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 <code>TriggerSample</code> 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.</p> <p>As a special case, setting <code>TriggerSample</code> 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.</p>	Yes
TriggerScope	<p>If <code>TriggerMode</code> 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 <code>TriggerScope</code> to the scope index of the master scope.</p>	Yes
TriggerSignal	<p>If <code>TriggerMode</code> 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 <code>Signal</code>.</p>	Yes
TriggerSlope	<p>If <code>TriggerMode</code> is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.</p>	Yes
Type	<p>Determines whether the scope is displayed on the development computer or on the target computer. Values are 'Host', 'Target', and 'File'.</p> <p>Property <code>Type</code> is set only once, when the scope is created on the target computer.</p>	No

## **xpctarget.xpcsc.remsignal**

Remove signals from scope represented by scope object (not recommended)

### **Syntax**

#### **MATLAB command line**

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

#### **Target command line**

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

### **Arguments**

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



- Method `xpctarget.xpcsc.remsignal` will be removed in a future release. Use methods `SimulinkRealTime.targetScope.remsignal`, `SimulinkRealTime.hostScope.remsignal`, and `SimulinkRealTime.fileScope.remsignal` instead.
- 

## 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])
```

## See Also

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

## **xpctarget.xpcsc.start (scope object)**

Start execution of scope on target computer (not recommended)

### **Syntax**

#### **MATLAB command line**

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

#### **Target computer command line**

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

### **Arguments**

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

### **Description**

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

---

**Note:** Method `xpctarget.xpcsc.start (scope object)` will be removed in a future release. Use methods `SimulinkRealTime.targetScope.start`, `SimulinkRealTime.hostScope.start`, and `SimulinkRealTime.fileScope.start` instead.

---

## Examples

Start one scope with the scope object `sc1`.

```
sc1 = getscope(tg,1)
start(sc1)
```

or type

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

Start two scopes.

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

or type

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

or type

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

Start all scopes:

```
allscopes = getscope(tg)
start(allscopes)
```

or type

```
start(getscope(tg))
```

## See Also

`xpctarget.xpc.stop (real-time application object)` |  
`xpctarget.xpc.getscope` | `xpctarget.xpcsc.stop (scope object)`

## xpctarget.xpcsc.stop (scope object)

Stop execution of scope on target computer (not recommended)

### Syntax

#### MATLAB command line

```
stop(scope_object_vector)
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 <code>getscope</code> , which returns a <code>scope_object</code> 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.

---

**Note:** Method `xpctarget.xpcsc.stop (scope object)` will be removed in a future release. Use methods `SimulinkRealTime.targetScope.stop`,

SimulinkRealTime.hostScope.stop, and SimulinkRealTime.fileScope.stop instead.

---

## Examples

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

```
stop(sc1)
```

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

```
allscopes = getscope(tg)
stop(allscopes)
```

or type

```
stop(getscope(tg))
```

## See Also

`xpctarget.xpc.stop` (real-time application object) |  
`xpctarget.xpc.getscope` | `xpctarget.xpc.start` (real-time application object) | `xpctarget.xpcsc.start` (scope object)

## xpctarget.xpcsc.trigger

Software-trigger start of data acquisition for scopes (not recommended)

### Syntax

```
trigger(scope_object_vector)
```

### Arguments

<code>scope_object_vector</code>	Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form [ <code>scope_object1</code> , <code>scope_object2</code> ], or the target object method <code>getscope</code> , which returns a <code>scope_object</code> 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:** Method `xpctarget.xpcsc.trigger` will be removed in a future release. Use methods `SimulinkRealTime.targetScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.fileScope.trigger` instead.

---

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

## xpctarget.xpcschoost Class

Control and access properties of host scopes (not recommended)

### Description

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

---

**Note:** Class `xpctarget.xpcschoost` will be removed in a future release. Use class `SimulinkRealTime.hostScope` instead.

---

### Methods

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

Method	Description
<code>xpctarget.xpcsc.addsigr</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.remsigr</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	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



Property	Description	Writable
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <code>TriggerMode</code> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>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.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
TriggerLevel	If <code>TriggerMode</code> 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	<p>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.</p> <p>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.</p>	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
Type	<p>Determines whether the scope is displayed on the development computer or on the target computer. Values are 'Host', 'Target', and 'File'.</p> <p>Property Type is set only once, when the scope is created on the target computer.</p>	No

These properties are specific to class xpcschoost.

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

## xpctarget.xpcsc Class

Control and access properties of target scopes (not recommended)

### 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.

---

**Note:** Class `xpctarget.xpcsc` will be removed in a future release. Use class `SimulinkRealTime.targetScope` instead.

---

### Methods

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

Method	Description
<code>xpctarget.xpcsc.addsig</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.remsig</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	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

Property	Description	Writable
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <code>TriggerMode</code> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>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.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
TriggerLevel	If <code>TriggerMode</code> 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	<p>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.</p> <p>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.</p>	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
Type	<p>Determines whether the scope is displayed on the development computer or on the target computer. Values are 'Host', 'Target', and 'File'.</p> <p>Property Type is set only once, when the scope is created on the target computer.</p>	No

These properties are specific to class `xpcsctg`.

Property	Description	Writeable
DisplayMode	<p>For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For host or file scopes, this parameter has no effect.</p> <p>.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes
Mode	<p><b>Note:</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"> <li>• For target scopes, use DisplayMode.</li> <li>• For file scopes, use WriteMode.</li> <li>• For host scopes, this parameter has no effect.</li> </ul>	Yes
YLimit	<p>Minimum and maximum <i>y</i>-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

## xpctargetping

Tests communication between development and target computers (not recommended)

### Syntax

```
xpctargetping
```

```
xpctargetping target_computer_name
```

### Description

Returns **success** if the Simulink Real-Time kernel is loaded and running, and communication is working between the development and target computers. Otherwise, returns **failed**.

---

**Note:** Command `xpctargetping` will be removed in a future release. Use command `slrtpingtarget` or method `SimulinkRealTime.target.ping` instead.

---

`xpctargetping` without an argument returns **success** if the development computer and the default target computer can communicate using the settings for that computer. Otherwise, returns **failed**.

`xpctargetping target_computer_name` returns **success** if the development computer can communicate with target computer `target_computer_name` using the settings for that computer. Otherwise, returns **failed**.



## Examples

### Check communication with default target computer

```
xpctargetping
```

### Check communication with specified target computer

```
xpctargetping TargetPC1
```

## Input Arguments

### **target\_computer\_name** — Name of specific target computer

TargetPC1 | TargetPC2 | ...

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

When using function form, enclose the argument (`target_computer_name`,) in single quotes ('TargetPC1').

Example: TargetPC1

Data Types: char

## xpctargetspy

Open Simulink Real-Time display window on development computer (not recommended)

### Syntax

```
xpctargetspy  
xpctargetspy(target_object)  
xpctargetspy('target_object_name')
```

### Arguments

<code>target_object</code>	Variable name to reference the target object.
<code>target_object_name</code>	Target object name as specified in the Simulink Real-Time Explorer.

### Description

This graphical user interface (GUI) allows you to upload displayed data from the target computer. By default, `xpctargetspy` opens a Simulink Real-Time real-time display 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`.

---

**Note:** Command `xpctargetspy` will be removed in a future release. Use command method `SimulinkRealTime.target.viewTargetScreen` instead.

---

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 specified a target computer object in the Simulink Real-Time Explorer, you can use the following syntax.

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

Then, use the following syntax.

```
xpctargetspy(target_object)
```

The behavior of `xpctargetspy` 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 Simulink Real-Time real-time display window and right-click to select **Update Simulink Real-Time Target Screen**.
- If `TargetScope` is disabled, text output is transferred once every second to the host and displayed in the window.

## Examples

To open the Simulink Real-Time real-time display window for the default target computer, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the Simulink Real-Time real-time display window for target computer 'TargetPC1' in the MATLAB window, type

```
tg1=xpctarget.xpc('TargetPC1');  
xpctargetspy(tg1)
```

## xpctest

Test Simulink Real-Time installation (not recommended)

### Syntax

```
xpctest
xpctest('noreboot')
xpctest('-noreboot')
xpctest('target_name')
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 tests to check the basic functioning of Simulink Real-Time.

---

**Note:** Command `xpctest` will be removed in a future release. Use command `slrttest` instead.

---

xpctest tests the following functionality:

- Initiate communication between the development and target computers.
- Reboot the target computer to reset the target environment.
- Build a real-time application on the development computer.
- Download a real-time application to the target computer.

- Check communication between the development and target computers using commands.
- Execute a real-time application.
- Compare the results of a simulation and the real-time application run.

`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','noreboot')` or `xpctest('target_name','-noreboot')` 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')
```

## More About

- “Run Confidence Test on Configuration”
- “Test 1: Ping Using System Ping”

## xpcwwwenable

Disconnect target computer from current client application (not recommended)

### Syntax

```
xpcwwwenable  
xpcwwwenable('target_obj_name')
```

### Description

xpcwwwenable disconnects the real-time application from the MATLAB interface so you can connect to the Web browser.

---

**Note:** Command xpcwwwenable will be removed in a future release. Use method SimulinkRealTime.target.close instead.

---

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 real-time application on target\_obj\_name (for example 'TargetPC1') from the MATLAB interface.

# Simulink Real-Time API Reference for C

---

## dirStruct

Type definition for file system folder information structure

### Syntax

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

### Fields

<i>Name</i>	This value contains the name of the file or folder.
<i>Ext</i>	This value contains the file type of the element, if the element is a file ( <i>isDir</i> is 0). If the element is a folder ( <i>isDir</i> is 1), this field is empty.
<i>Day</i>	This value contains the day the file or folder was last modified.
<i>Month</i>	This value contains the month the file or folder was last modified.
<i>Year</i>	This value contains the year the file or folder was last modified.
<i>Hour</i>	This value contains the hour the file or folder was last modified.
<i>Min</i>	This value contains the minute the file or folder was last modified.
<i>isDir</i>	This value indicates if the element is a file (0) or folder (1). If it is a folder, <b>Bytes</b> 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`

## diskinfo

Type definition for file system disk information structure

### Syntax

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

### Fields

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

---

<i>FATType</i>	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of <i>Files</i> .
<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to <i>FreeClusters</i> .

## Description

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

## **See Also**

API function `xPCFSDiskInfo`

# fileinfo

Type definition for file information structure

## Syntax

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

## Fields

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

## Description

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

## See Also

`xPCFSFileInfo`

# lgmode

Type definition for logging options structure

## Syntax

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

## Fields

*mode*

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

*incrementvalue*

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

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

## Description

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

## See Also

API functions `xPCSetLogMode`, `xPCGetLogMode`

# scopedata

Type definition for scope data structure

## Syntax

```
typedef struct {
    int    number;
    int    type;
    int    state;
    int    signals[10];
    int    numsamples;
    int    decimation;
    int    triggermode;
    int    numprepostsamples;
    int    triggersignal;
    int    triggerscope;
    int    triggerscopesample;
    double triggerlevel;
    int    triggerslope;
} scopedata;
```

## Fields

<i>number</i>	The scope number.										
<i>type</i>	Determines whether the scope is displayed on the development computer or on the target computer. Values are one of the following: <table><tr><td>1</td><td>Host</td></tr><tr><td>2</td><td>Target</td></tr></table>	1	Host	2	Target						
1	Host										
2	Target										
<i>state</i>	Indicates the scope state. Values are one of the following: <table><tr><td>0</td><td>Waiting to start</td></tr><tr><td>1</td><td>Scope is waiting for a trigger</td></tr><tr><td>2</td><td>Data is being acquired</td></tr><tr><td>3</td><td>Acquisition is finished</td></tr><tr><td>4</td><td>Scope is stopped (interrupted)</td></tr></table>	0	Waiting to start	1	Scope is waiting for a trigger	2	Data is being acquired	3	Acquisition is finished	4	Scope is stopped (interrupted)
0	Waiting to start										
1	Scope is waiting for a trigger										
2	Data is being acquired										
3	Acquisition is finished										
4	Scope is stopped (interrupted)										

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



## 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`

# xPCAddScope

Create new scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scType</i>	Enter the type of scope.
<i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .

## Description

The `xPCAddScope` function creates a new scope on the target computer. For *scType*, scopes can be of type `host` or `target`, depending on the value of *scType*:

- `SCTYPE_HOST` for type `host`
- `SCTYPE_TARGET` for type `target`
- `SCTYPE_FILE` for type `file`

Constants for *scType* are defined in the header file `xpcapiconst.h` as `SCTYPE_HOST`, `SCTYPE_TARGET`, and `SCTYPE_FILE`.

Calling the `xPCAddScope` function with *scNum* having the number of an existing scope produces an error. Use `xPCGetScopes` to find the numbers of existing scopes.

## See Also

API functions `xPCScAddSignal`, `xPCScRemSignal`, `xPCRemScope`, `xPCSetScope`, `xPCGetScope`, `xPCGetScopes`

Target object method `SimulinkRealTime.target.addscope`

# xPCAverageTET

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 real-time application.

## Description

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

## See Also

API functions `xPCMaximumTET`, `xPCMinimumTET`

Property `AvgTET` of `SimulinkRealTime.target`

## xPCCloseConnection

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 communication type will be removed in a future release. Use TCP/IP instead.

---

### See Also

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

# xPCClosePort

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 communication type will be removed in a future release. Use TCP/IP instead.

---

## See Also

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

Target object method `SimulinkRealTime.target.close`

## xPCDeRegisterTarget

Delete target communication properties from Simulink Real-Time API library

### Prototype

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

### Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

### Description

The `xPCDeRegisterTarget` function causes the Simulink Real-Time API library to completely “forget” about the target communication properties. You use this at the end of a session in which you use `xPCOpenConnection` and `xPCCloseConnection` to connect and disconnect from the target without entering the properties each time. It works similarly to `xPCClosePort`, but does not close the connection to the target computer. Before calling this function, you must first call the function `xPCCloseConnection` to close the connection to the target computer. The combination of calling the `xPCCloseConnection` and `xPCDeRegisterTarget` functions has the same result as calling `xPCClosePort`.

### See Also

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

# xPCErrorMsg

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



# xPCFreeAPI

Unload Simulink Real-Time DLL

## Prototype

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

## Description

The `xPCFreeAPI` function unloads the Simulink Real-Time dynamic link library. You must execute this function once at the end of the application to unload the Simulink Real-Time API DLL. This frees the memory allocated to the functions. This function is defined in the file `xpcinitfree.c`. Link this file with your application.

## See Also

API functions `xPCInitAPI`, `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

# xPCFSCD

Change current folder on target computer to specified path

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dir</i>	Enter the path on the target computer to change to.

## Description

The `xPCFSCD` function changes the current folder on the target computer to the path specified in *dir*. Use the `xPCFSGetPWD` function to show the current folder of the target computer.

## See Also

API function `xPCFSGetPWD`

File object method `SimulinkRealTime.fileSystem.cd`

# xPCFSCloseFile

Close file on target computer

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

## Description

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

## See Also

API functions `xPCFSOpenFile`, `xPCFSReadFile`, `xPCFSWriteFile`

File object method `SimulinkRealTime.fileSystem.fclose`

# xPCFSDir

Get contents of specified folder on target computer

## Prototype

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

## Arguments

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

## Description

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

## See Also

API function `xPCFSDirSize`

File object method `SimulinkRealTime.fileSystem.dir`

## xPCFSDirItems

Get contents of specified folder on target computer

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the path on the target computer.
<i>dirs</i>	Enter the structure to contain the contents of the folder.
<i>numDirItems</i>	Enter the number of items in the folder.

### Description

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

### See Also

API functions `xPCFSDirStructSize`, `dirStruct`

File object method `SimulinkRealTime.fileSystem.dir`

# xPCFSDirSize

Return size of specified folder listing on target computer

## Prototype

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

## Arguments

- |             |  |
|-------------|--|
| <i>port</i> | Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> . |
| <i>path</i> | Enter the folder path on the target computer.  |

## Return

The `xPCFSDirSize` function returns the size, in bytes, of the specified folder listing. If this function detects an error, it returns -1.

## Description

The `xPCFSDirSize` function returns the size, in bytes, of the buffer required to list the folder contents on the target computer. Use this size as the *numbytes* parameter in the `xPCFSDir` function.

## See Also

API function `xPCFSDirItems`

File object method `SimulinkRealTime.fileSystem.dir`

## xPCFSDirStructSize

Get number of items in folder

### Prototype

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

### Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*path*            Enter the folder path on the target computer.

### Return

The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. If this function detects an error, it returns -1.

### Description

The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. Use this size as the *numDirItems* parameter in the `xPCFSDirItems` function.

### See Also

API function `xPCFSDir`

File object method `SimulinkRealTime.fileSystem.dir`

# xPCFSDiskInfo

Information about target computer file system

## Prototype

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

## Arguments

*port*

Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*driveletter*

Enter the drive letter of the file system for which you want information.

## Description

The `xPCFSDiskInfo` function returns disk information for the file system of the specified target computer drive, *driveletter*. This function returns this information in the `diskinfo` structure.

## See Also

API structure `SimulinkRealTime.fileSystem.diskinfo`



## xPCFSFileInfo

Return information for open file on target computer

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

### Description

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

### See Also

Structure `SimulinkRealTime.fileSystem.fileinfo`

## xPCFSGetError

Get text description for error number on target computer file system

### Prototype

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

### Arguments

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

### Description

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

## xPCFSGetFileSize

Return size of file on target computer

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

### Return

Return the size of the specified file in bytes. If this function detects an error, it returns -1.

### Description

The `xPCFSGetFileSize` function returns the size, in bytes, of the file associated with *fileHandle* on the target computer. *fileHandle* is the handle of a file previously opened by the `xPCFSOpenFile` function.

### See Also

API functions `xPCFSOpenFile`, `xPCFSReadFile`

File object methods `SimulinkRealTime.fileSystem.fopen` and `SimulinkRealTime.fileSystem.fread`

# xPCFSGetPWD

Get current folder of target computer

## Prototype

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

## Arguments

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

## Description

The `xPCFSGetPWD` function places the path of the current folder on the target computer in *pwd*, which must be allocated by the caller.

## See Also

File object method `SimulinkRealTime.fileSystem.pwd`

## xPCFSMKDIR

Create new folder on target computer

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of the folder to create on the target computer.

### Description

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

### See Also

API function `xPCFSGetPWD`

File object method `SimulinkRealTime.fileSystem.mkdir`

# xPCFSOpenFile

Open file on target computer

## Prototype

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

## Arguments

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

## Return

The `xPCFSOpenFile` function returns the file handle for the opened file. If function detects an error, it returns -1.

## Description

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

## See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSReadFile`, `xPCFSWriteFile`

File object methods `SimulinkRealTime.fileSystem.fclose`,  
`SimulinkRealTime.fileSystem.filetable`,  
`SimulinkRealTime.fileSystem.fwrite` `SimulinkRealTime.fileSystem.fopen`  
and `SimulinkRealTime.fileSystem.fread`

# xPCFSReadFile

Read open file on target computer

## Prototype

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

## Arguments

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

## Description

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

## See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSWriteFile`



File object methods `SimulinkRealTime.fileSystem.fopen` and  
`SimulinkRealTime.fileSystem.fread`

# xPCFSRemoveFile

Remove file from target computer

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of a file on the target computer.

## Description

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

## See Also

File object method `SimulinkRealTime.fileSystem.removefile`

## xPCFSRMDIR

Remove folder from target computer

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of a folder on the target computer.

### Description

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

### See Also

File object method `SimulinkRealTime.fileSystem.rmdir`

# xPCFSScGetFilename

Get name of file for scope

## Prototype

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

## Arguments

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

## Return

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

## Description

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

## See Also

API function `xPCFSScSetFilename`

Property `Filename` of `SimulinkRealTime.fileSystem`

# xPCFSScGetWriteMode

Get write mode of file for scope

## Prototype

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

## Arguments

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

## Return

Returns the number indicating the write mode. Values are

- |   |  |
|---|--|
| 0 | Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact). |
| 1 | Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.   |

## Description

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

## See Also

API function `xPCFSScSetWriteMode`

Property WriteMode of SimulinkRealTime.fileSystem

## xPCFSScGetWriteSize

Get block write size of data chunks

### Prototype

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

### Arguments

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

### Return

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

### Description

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

### See Also

API function `xPCFSScSetWriteSize`

Property `WriteSize` of `SimulinkRealTime.fileSystem`

## xPCFSScSetFilename

Specify name for file to contain signal data

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>filename</i>	Enter the name of a file to contain the signal data.

### Description

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

### See Also

API function `xPCFSScGetFilename`

Property `Filename` of `SimulinkRealTime.fileSystem`



## xPCFSScSetWriteMode

Specify when file allocation table entry is updated

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeMode</i>	Enter an integer for the write mode: 0      Enables lazy write mode 1      Enables commit write mode

### Description

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

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

### See Also

API function `xPCFSScGetWriteMode`

Property WriteMode of SimulinkRealTime.fileSystem

## xPCFSScSetWriteSize

Specify that memory buffer collect data in multiples of write size

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeSize</i>	Enter the block size, in bytes, of the data chunks.

### Description

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

### See Also

API function `xPCFSScGetWriteSize`

Property `WriteSize` of `SimulinkRealTime.fileSystem`

## xPCFSWriteFile

Write to file on target computer

### Prototype

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

### Arguments

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

### Description

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

### See Also

API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSReadFile`

## xPCGetAPIVersion

Get version number of Simulink Real-Time API

### Prototype

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

### Return

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

### Description

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

### See Also

API function `xPCGetTargetVersion`

# xPCGetAppName

Return real-time application name

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>model_name</i>	The <code>xPCGetAppName</code> function copies the real-time application name string into the buffer pointed to by <i>model_name</i> . <i>model_name</i> is then returned. You can later use <i>model_name</i> in a function such as <code>printf</code> .

Note that the maximum size of the buffer is 256 bytes. To reserve enough space for the application name string, allocate a buffer of size 256 bytes.

## Return

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

## Description

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

# xPCGetEcho

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 real-time application, changes to parameters, and changes to scope signals.



## See Also

API function xPCSetEcho

# xPCGetExecTime

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

## Description

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

## See Also

API functions `xPCSetStopTime`, `xPCGetStopTime`

Property `ExecTime` of `SimulinkRealTime.target`

## xPCGetLastError

Return constant of last error

### Prototype

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

### Return

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

### Description

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

### See Also

API functions `xPCErrorMsg`, `xPCSetLastError`

# xPCGetLoadTimeOut

Return timeout value for communication between development and target computers

## Prototype

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

## Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

## Return

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

## Description

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

For example, when you load a new real-time application onto the target computer, the function `xPCLoadApp` waits for a certain amount of time before checking to see if the initialization of the real-time application is complete. In the case where initialization of the real-time 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

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

## See Also

API function `xPCSetLogMode`

API structure `lgmode`

## xPCGetNumOutputs

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 real-time application. If the function detects an error, it returns -1.

### Description

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

### See Also

API functions `xPCGetOutputLog`, `xPCGetNumStates`, `xPCGetStateLog`

# xPCGetNumParams

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 real-time application. If the function detects an error, it returns -1.

## Description

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

## See Also

API functions `xPCGetParamIdx`, `xPCSetParam`, `xPCGetParam`, `xPCGetParamName`, `xPCGetParamDims`

Property `NumParameters` of `SimulinkRealTime.target`



## xPCGetNumScopes

Return number of scopes added to real-time 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 real-time application. If the function detects an error, it returns -1.

### Description

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

# xPCGetNumScSignals

Returns number of signals added to specific scope

## Prototype

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

## Arguments

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

## Return

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

## Description

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

# xPCGetNumSignals

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

## Description

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

## See Also

API functions `xPCGetSignalIdx`, `xPCGetSignal`, `xPCGetSignals`, `xPCGetSignalName`, `xPCGetSignalWidth`

Property `NumSignals` of `SimulinkRealTime.target`

# xPCGetNumStates

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 real-time application. If the function detects an error, it returns -1.

## Description

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

## See Also

API functions `xPCGetStateLog`, `xPCGetNumOutputs`, `xPCGetOutputLog`

Property `StateLog` of `SimulinkRealTime.target`

# xPCGetOutputLog

Copy output log data to array

## Prototype

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

## Arguments

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

## Description

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

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

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

## See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Target object method `SimulinkRealTime.target.getlog`

Property `OutputLog` of `SimulinkRealTime.target`

## xPCGetParam

Get parameter value and copy it to array

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Enter the index for a parameter.
<i>paramValue</i>	The function returns a parameter value as an array of doubles.

### Description

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

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

### See Also

API functions `xPCSetParam`, `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetNumParams`

`SimulinkRealTime.target.getparamid`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`

# xPCGetParamDims

Get row and column dimensions of parameter

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Parameter index.
<i>dimension</i>	Dimensions (row, column) of a parameter.

## Description

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

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

## See Also

API functions `xPCGetParamIdx`, `xPCGetParamName`, `xPCSetParam`, `xPCGetParam`, `xPCGetNumParams`

`SimulinkRealTime.target.getparamid`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`



# xPCGetParamIdx

Return parameter index

## Prototype

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

## Arguments

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

## Return

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

## Description

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

## See Also

API functions `xPCGetParamDims`, `xPCGetParamName`, `xPCGetParam`

`SimulinkRealTime.target.getparamid`

Properties ShowParameters and Parameters of `SimulinkRealTime.target`

# xPCGetParamName

Get name of parameter

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Enter a parameter index.
<i>blockName</i>	String with the full block path generated by Simulink Coder.
<i>paramName</i>	Name of a parameter for a specific block.

## Description

The `xPCGetParamName` function gets the parameter name and block name for a parameter with the index *paramIdx*. The block path and name are returned and stored in *blockName*, and the parameter name is returned and stored in *paramName*. You must allocate enough space for both *blockName* and *paramName*. If the *paramIdx* is invalid, `xPCGetLastError` returns nonzero, and the strings are unchanged. Get the parameter index from the function `xPCGetParamIdx`.

## See Also

API functions `xPCGetParam`, `xPCGetParamDims`, `xPCGetParamIdx`

Properties `ShowParameters` and `Parameters` of `SimulinkRealTime.target`

# xPCGetSampleTime

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

## Description

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

## See Also

API function `xPCSetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

# xPCGetScope

Get and copy scope data to structure

## Prototype

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

## Arguments

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

## Return

The `xPCGetScope` function returns a structure of type `scopedata`.

## Description

---

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

---

The `xPCGetScope` function gets properties of a scope with `scNum` and copies the properties into a structure with type `scopedata`. You can use this function in conjunction with `xPCSetScope` to change several properties of a scope at one time. See `scopedata` for a list of properties. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCSetScope`, `scopedata`

Target object method `SimulinkRealTime.target.getscope`

## xPCGetScopeList

Get and copy list of scope numbers

### Prototype

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

### Arguments

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

### Description

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

---

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

---

# xPCGetScopes

Get and copy list of scope numbers

## Prototype

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

## Arguments

- |             |  |
|-------------|--|
| <i>port</i> | Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> . |
| <i>data</i> | List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers and terminated by -1.       |

## Description

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

---

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

---

## See Also

API functions `xPCSetScope`, `xPCGetScope`, `xPCScGetSignals`

Property Scopes of `SimulinkRealTime.target`



## xPCGetSessionTime

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

### Prototype

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

### Arguments

*port*                      Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

### Return

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

### Description

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

# xPCGetSignal

Return value of signal

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigNum</i>	Enter a signal number.

## Return

The `xPCGetSignal` function returns the current value of signal *sigNum*. If the function detects an error, it returns -1.

## Description

The `xPCGetSignal` function returns the current value of a signal. For vector signals, use `xPCGetSignals` rather than call this function multiple times. Use the `xPCGetSignalIdx` function to get the signal number.

## See Also

API function `xPCGetSignals`

Property `Signals` of `SimulinkRealTime.target`

# xPCGetSignalIdx

Return index for signal

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigName</i>	Enter a signal name.

## Return

The `xPCGetSignalIdx` function returns the index for the signal with name *sigName*. If the function detects an error, it returns -1.

## Description

The `xPCGetSignalIdx` function returns the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file `model_namebio.m` in the generated code, where *model\_name* is the name of the model. The creator of the application should already know the signal name.

## See Also

API functions `xPCGetSignalName`, `xPCGetSignalWidth`, `xPCGetSignal`, `xPCGetSignals`

Target object method `SimulinkRealTime.target.getsignalid`

## xPCGetSigIdxfromLabel

Return array of signal indices

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigLabel</i>	String with the name of a signal label.
<i>sigIds</i>	Return array of signal indices.

### Return

If `xPCGetSigIdxfromLabel` finds a signal, it fills an array *sigIds* with signal indices and returns 0. If it finds no signal, it returns -1.

### Description

The `xPCGetSigIdxfromLabel` function returns in *sigIds* the array of signal indices for signal *sigName*. This function assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

*sigIds* must be large enough to contain the array of indices. You can use the `xPCGetSigLabelWidth` function to get the required amount of memory to be allocated by the *sigIds* array.

## See Also

API functions `xPCGetSignalLabel`, `xPCGetSigLabelWidth`

## xPCGetSignalLabel

Copy label of signal to character array

### Prototype

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

### Arguments

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

### Return

The `xPCGetSignalLabel` function returns the label of the signal.

### Description

The `xPCGetSignalLabel` function copies and returns the signal label, including the block path, of a signal with *sigIdx*. The result is stored in *sigLabel*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigLabel* is unchanged. The function returns *sigLabel*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index. Signal labels must be unique.

This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

## See Also

API functions `xPCGetSigIdxfromLabel`, `xPCGetSigLabelWidth`

# xPCGetSigLabelWidth

Return number of elements in signal

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigName</i>	String with the name of a signal.

## Return

The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. If the function detects an error, it returns -1.

## Description

The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. This function assumes that you have labeled the signal for which you request the elements (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the Simulink Real-Time software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

## See Also

API functions `xPCGetSigIdxfromLabel`, `xPCGetSignalLabel`



# xPCGetSignalName

Copy name of signal to character array

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter a signal index.
<i>sigName</i>	String with the name of a signal.

## Return

The `xPCGetSignalName` function returns the name of the signal.

## Description

The `xPCGetSignalName` function copies and returns the signal name, including the block path, of a signal with *sigIdx*. The result is stored in *sigName*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigName* is unchanged. The function returns *sigName*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index.

## See Also

API functions `xPCGetSignalIdx`, `xPCGetSignalWidth`, `xPCGetSignal`, `xPCGetSignals`

Properties `ShowSignals` and `Signals` of `SimulinkRealTime.target`

## xPCGetSignals

Return vector of signal values

### Prototype

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

### Arguments

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

### Return

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

### Description

The `xPCGetSignals` function is the vector version of the function `xPCGetSignal`. This function returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step (for that, define a scope of type `SCTYPE_HOST` and use `xPCScGetData`). `xPCGetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result. However, the `xPCGetSignals` function is faster, and the signal values are more likely to be spaced closely together. The signals are converted to doubles regardless of the actual data type of the signal.

For *signals*, the list you provide should be stored in an integer array. Get the signal numbers with the function `xPCGetSignalIdx`.

## See Also

API function `xPCGetSignal`, `xPCGetSignalIdx`

## Example

To reference signal vector data rather than scalar values, pass a vector of indices for the signal data. For example:

```

/*****/

/* Assume a signal of width 10, with the blockpath
 * mySubsys/mySignal and the signal index s1.
 */

int i;
int sigId[10];
double sigVal[10]; /* Signal values are stored here */

/* Get the ID of the first signal */
sigId[0] = xPCGetSignalIdx(port, "mySubsys/mySignal/s1");

if (sigId[0] == -1) {
    /* Handle error */
}

for (i = 1; i < 10; i++) {
    sigId[i] = sigId[0] + i;
}

xPCGetSignals(port, 10, sigId, sigVal);
/* If no error, sigVal should have the signal values */

/*****/

```

To repeatedly get the signals, repeat the call to `xPCGetSignals`. If you do not change `sigID`, you only need to call `xPCGetSignalIdx` once.

# xPCGetSignalWidth

Return width of signal

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter the index of a signal.

## Return

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

## Description

The `xPCGetSignalWidth` function returns the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector again. A signal's width is the number of signals in the vector.

## See Also

API functions `xPCGetSignalIdx`, `xPCGetSignalName`, `xPCGetSignal`, `xPCGetSignals`

## xPCGetStateLog

Copy state log values to array

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>state_id</i>	Enter a state identification number.
<i>state_data</i>	The log is stored in <i>state_data</i> , whose allocation is the responsibility of the caller.

### Description

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

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

## See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumStates`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

`SimulinkRealTime.target.getlog`

Property `StateLog` of `SimulinkRealTime.target`

# xPCGetStopTime

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 real-time application. If the function detects an error, it returns `-10.0`. If the stop time is infinity (run forever), this function returns `-1.0`.

## Description

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

## See Also

API function `xPCSetStopTime`

Property `StopTime` of `SimulinkRealTime.target`

# xPCGetTargetVersion

Get Simulink Real-Time kernel version

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ver</i>	The version is stored in <i>ver</i> .

## Description

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

## See Also

`xPCGetAPIVersion`



## xPCGetTETLog

Copy TET log to array

### Prototype

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

### Arguments

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

### Description

The `xPCGetTETLog` function gets the task execution time (TET) log. It then copies the log into *TET\_data*. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

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

### See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTimeLog`

`SimulinkRealTime.target.getlog`

Property TETLog of `SimulinkRealTime.target`

# xPCGetTimeLog

Copy time log to array

## Prototype

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

## Arguments

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

## Description

The `xPCGetTimeLog` function gets the time log and copies the log into *time\_data*. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

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

## See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

`SimulinkRealTime.target.getlog`

Property TimeLog of `SimulinkRealTime.target`

# xPCInitAPI

Initialize Simulink Real-Time DLL

## Prototype

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

## Return

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

## Description

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

## See Also

API functions `xPCFreeAPI`, `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

# xPCIsAppRunning

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

## Description

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

## See Also

API function `xPCIsOverloaded`

Property Status of `SimulinkRealTime.target`

# xPCIsOverloaded

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

## Description

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

## See Also

API function `xPCIsAppRunning`

Property `CPUOverload` of `SimulinkRealTime.target`

# xPCIsScFinished

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`



# xPCLoadApp

Load real-time application onto target computer

## Prototype

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

## Arguments

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

## Description

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

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

## See Also

API functions `xPCStartApp`, `xPCStopApp`, `xPCUnloadApp`, `xPCSetLoadTimeOut`, `xPCGetLoadTimeOut`

Target object method `SimulinkRealTime.target.load`

# xPCLoadParamSet

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

## See Also

API function xPCSaveParamSet

# xPCMaxLogSamples

Return maximum number of samples that can be in log buffer

## Prototype

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

## Arguments

*port*          Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

## Return

The `xPCMaxLogSamples` function returns the total number of samples. If the function detects an error, it returns -1.

## Description

The `xPCMaxLogSamples` function returns the total number of samples that can be returned in the logging buffers.

## See Also

API functions `xPCNumLogSamples`, `xPCNumLogWraps`, `xPCGetStateLog`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Property `MaxLogSamples` of `SimulinkRealTime.target`

## xPCMaximumTET

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

### See Also

API functions `xPCMinimumTET`, `xPCAverageTET`

Property `MaxTET` of `SimulinkRealTime.target`

# xPCMinimumTET

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

## See Also

API functions `xPCMaximumTET`, `xPCAverageTET`

Property `MinTET` of `SimulinkRealTime.target`

## xPCNumLogSamples

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

### See Also

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

# xPCNumLogWraps

Return number of times log buffer wraps

## Prototype

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

## Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

## Return

The `xPCNumLogWraps` function returns the number of times the log buffer wraps. If the function detects an error, it returns -1.

## Description

The `xPCNumLogWraps` function returns the number of times the log buffer wraps.

## See Also

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

Property `NumLogWraps` of `SimulinkRealTime.target`



# xPCOpenConnection

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

Open RS-232 connection to Simulink Real-Time system

## Prototype

```
int xPCOpenSerialPort(int comPort, int baudRate);
```

## Arguments

<i>comPort</i>	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
<i>baudRate</i>	<i>baudRate</i> must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

## Return

The `xPCOpenSerialPort` function returns the port value for the connection. If the function detects an error, it returns -1.

## Description

The `xPCOpenSerialPort` function initiates an RS-232 connection to a Simulink Real-Time system. It returns the port value for the connection. Be sure to pass this value to all the Simulink Real-Time API functions that require a port value.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

---

**Note:** RS-232 communication type will be removed in a future release. Use TCP/IP instead.

---

## See Also

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

# xPCOpenTcpIpPort

Open TCP/IP connection to Simulink Real-Time system

## Prototype

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

## Arguments

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

## Return

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

## Description

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

## See Also

API functions `xPCOpenSerialPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCTargetPing`

# xPCReboot

Reboot target computer

## Prototype

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

## Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

## Description

The `xPCReboot` function reboots the target computer. This function returns nothing. This function does not close the connection to the target computer. You should either explicitly close the port or call `xPCReOpenPort` once the target computer has rebooted.

## See Also

API function `xPCReOpenPort`

Target object method `SimulinkRealTime.target.reboot`

# xPCReOpenPort

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

Register target with Simulink Real-Time API library

## Prototype

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

## Arguments

*commType* Specify the communication type (TCP/IP or RS-232) between the development and target computers.

---

**Note:** RS-232 communication type 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

Remove scope

## Prototype

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

## Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum*          Enter the scope number.

## Description

The `xPCRemScope` function removes the scope with number *scNum*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, see `xPCGetScopes`. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCAddScope`, `xPCScRemSignal`, `xPCGetScopes`

Target object method `SimulinkRealTime.target.remscope`

# xPCSaveParamSet

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

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

## See Also

API function `xPCLoadParamSet`

# xPCScAddSignal

Add signal to scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

## Description

The `xPCScAddSignal` function adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScGetSignals` to get a list of the signals already present. Use the function `xPCGetScope` to get the scope number. Use the `xPCGetSignalIdx` function to get the signal number.

## See Also

API functions `xPCScRemSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`

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

## xPCScGetAutoRestart

Scope autorestart status

### Prototype

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

### Arguments

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

### Return

The `xPCScGetAutoRestart` function returns the autorestart flag value of scope *scNum*. If the function detects an error, it returns -1.

### Description

The `xPCScGetAutoRestart` function gets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (0) or enabled (1).

### See Also

API functions `xPCScSetAutoRestart`

## xPCScGetData

Copy scope data to array

### Prototype

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

### Arguments

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

### Description

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

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

### See Also

API functions `xPCGetScope`, `xPCScGetState`, `xPCScGetSignals`

Property Data of `SimulinkRealTime.hostScope`

# xPCScGetDecimation

Return decimation of scope

## Prototype

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

## Arguments

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

## Return

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

## Description

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

## See Also

API function `xPCScSetDecimation`

Property `Decimation` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScGetNumPrePostSamples

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

### Prototype

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

### Arguments

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

### Return

The `xPCScGetNumPrePostSamples` function returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this function returns the minimum integer value (-2147483647 - 1).

### Description

The `xPCScGetNumPrePostSamples` function gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples. Use the `xPCGetScope` function to get the scope number.

### See Also

API function `xPCScSetNumPrePostSamples`

Property `NumPrePostSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`



# xPCScGetNumSamples

Get number of samples in one data acquisition cycle

## Prototype

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

## Arguments

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

## Return

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

## Description

The `xPCScGetNumSamples` function gets the number of samples in one data acquisition cycle for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

## See Also

API function `xPCScSetNumSamples`

Property `NumSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScGetNumSignals

Get number of signals in scope

### Prototype

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

### Arguments

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

### Return

The `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

Copy list of signals to array

## Prototype

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

## Arguments

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

## Description

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

---

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

---

# xPCScGetSignals

Copy list of signals to array

## Prototype

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

## Arguments

<i>port</i>	Value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	Integer array allocated by the caller as a list containing the signal identifiers, terminated by -1.

## Description

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

---

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

---

## See Also

API functions `xPCScGetData`, `xPCGetScopes`

Scope object property `Signals`

## xPCScGetStartTime

Get start time for last data acquisition cycle

### Prototype

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

### Arguments

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

### Return

The `xPCScGetStartTime` function returns the start time for the last data acquisition cycle of a scope. If the function detects an error, it returns -1.

### Description

The `xPCScGetStartTime` function gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`. Use the `xPCGetScope` function to get the scope number.

### See Also

API functions `xPCScGetNumSamples`, `xPCScGetDecimation`

## xPCScGetState

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.

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

## See Also

API functions `xPCScStart`, `xPCScStop`

Scope object property `Status`

# xPCScGetTriggerLevel

Get trigger level for scope

## Prototype

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

## Arguments

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

## Return

The `xPCScGetTriggerLevel` function returns the scope trigger level. If the function detects an error, it returns -1.

## Description

The `xPCScGetTriggerLevel` function gets the trigger level for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Property `TriggerLevel` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`



# xPCScGetTriggerMode

Get trigger mode for scope

## Prototype

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

## Arguments

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum* Enter the scope number.

## Return

The `xPCScGetTriggerMode` function returns the scope trigger mode. If the function detects an error, it returns -1.

## Description

The `xPCScGetTriggerMode` function gets the trigger mode for scope *scNum*. Use the `xPCGetScope` function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.

Constant	Value	Description
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerMode` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScGetTriggerScope

Get trigger scope

## Prototype

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

## Arguments

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

## Return

The `xPCScGetTriggerScope` function returns a trigger scope. If the function detects an error, it returns -1.

## Description

The `xPCScGetTriggerScope` function gets the trigger scope for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Property `TriggerScope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScGetTriggerScopeSample

Get sample number for triggering scope

### Prototype

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

### Arguments

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

### Return

The `xPCScGetTriggerScopeSample` function returns a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the function detects an error, it returns `INT_MIN (-2147483647 - 1)`.

### Description

The `xPCScGetTriggerScopeSample` function gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

### See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`,

xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope,  
xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScSetTriggerScopeSample

Property TriggerSample of SimulinkRealTime.fileScope,  
SimulinkRealTime.hostScope, and SimulinkRealTime.targetScope

# xPCScGetTriggerSignal

Get trigger signal for scope

## Prototype

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

## Arguments

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

## Return

The `xPCScGetTriggerSignal` function returns the scope trigger signal. If the function detects an error, it returns -1.

## Description

The `xPCScGetTriggerSignal` function gets the trigger signal for scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope.

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerSignal` of `SimulinkRealTime.fileScope`,  
`SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScGetTriggerSlope

Get trigger slope for scope

### Prototype

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

### Arguments

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

### Return

The `xPCScGetTriggerSlope` function returns the scope trigger slope. If the function detects an error, it returns -1.

### Description

The `xPCScGetTriggerSlope` function gets the trigger slope of scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope. Use the constants defined in `xpcapiconst.h` to interpret the trigger slope. These constants have the following meanings:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger slope must be rising when the signal crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger slope must be falling when the signal crosses the trigger value.



## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerSlope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScGetType

Get type of scope

## Prototype

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

## Arguments

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

## Return

The `xPCScGetType` function returns the scope type. If the function detects an error, it returns -1.

## Description

The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the development computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCAddScope`, `xPCRemScope`

Property Type of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`,  
and `SimulinkRealTime.targetScope`

# xPCScRemSignal

Remove signal from scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

## Description

The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCGetScopes` to determine the existing scopes, and use `xPCScGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

## See Also

API functions `xPCScAddSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`, `xPCScGetSignals`, `xPCScGetState`

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

# xPCScSetAutoRestart

Scope autorestart status

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>autorestart</i>	Enter value to enable (1) or disable (0) scope autorestart.

## Description

The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to 0 or 1. 0 disables the flag, 1 enables it. Use this function only when the scope is stopped.

## See Also

API functions `xPCScGetAutoRestart`

# xPCScSetDecimation

Set decimation of scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>decimation</i>	Enter an integer for the decimation.

## Description

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

## See Also

API functions `xPCScGetDecimation`, `xPCScGetState`

Property `Decimation` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScSetNumPrePostSamples

Set number of pre- or posttriggering samples before triggering scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>prepost</i>	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

## Description

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

## See Also

API functions `xPCScGetNumPrePostSamples`, `xPCScGetState`

Property `NumPrePostSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScSetNumSamples

Set number of samples in one data acquisition cycle

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>samples</i>	Enter the number of samples you want to acquire in one cycle.

## Description

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

## See Also

API functions `xPCScGetNumSamples`, `xPCScGetState`

Property `NumSamples` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`



# xPCScSetTriggerLevel

Set trigger level for scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>level</i>	Value for a signal to trigger data acquisition with a scope.

## Description

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

## See Also

API functions `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Property `TriggerLevel` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScSetTriggerMode

Set trigger mode of scope

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Trigger mode for a scope.

### Description

The `xPCScSetTriggerMode` function sets the trigger mode of scope `scNum` to `mode`. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

## See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScGetTriggerMode`, `xPCScGetState`

Methods `SimulinkRealTime.fileScope.trigger`, `SimulinkRealTime.hostScope.trigger`, and `SimulinkRealTime.targetScope.trigger`

Property `TriggerMode` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScSetTriggerScope

Select scope to trigger another scope

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigScope</i>	Enter the scope number of the scope used for a trigger.

### Description

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

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

### See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Property `TriggerScope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScSetTriggerScopeSample

Set sample number for triggering scope

## Prototype

```
void xPCScSetTriggerScopeSample(int port, int scNum, int trigScSamp);
```

## Arguments

- port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
- scNum* Enter the scope number.
- trigScSamp* Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.

## Description

The `xPCScSetTriggerScopeSample` function sets the number of samples (*trigScSamp*) a triggering scope acquires before it triggers a second scope (*scNum*). Use the `xPCGetScopes` function to get a list of scopes.

For meaningful results, set *trigScSamp* between `-1` and `(nSamp - 1)`. *nSamp* is the number of samples in one data acquisition cycle for the triggering scope. If you specify too large a value, the scope is never triggered.

If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, enter a value of `-1` for *trigScSamp*.

## See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`,

`xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`,  
`xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetTriggerScopeSample`

Property `TriggerSample` of `SimulinkRealTime.fileScope`,  
`SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCScSetTriggerSignal

Select signal to trigger scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSig</i>	Enter a signal number.

## Description

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

## See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Property `TriggerSignal` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScSetTriggerSlope

Set slope of signal that triggers scope

### Prototype

```
void xPCScSetTriggerSlope(int port, int scNum, int trigSlope);
```

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSlope</i>	Enter the slope mode for the signal that triggers the scope.

### Description

The `xPCScSetTriggerSlope` function sets the trigger slope of scope *scNum* to *trigSlope*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to set the trigger slope:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger signal value must be rising when it crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger signal value must be falling when it crosses the trigger value.



## See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Property `TriggerSlope` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScSoftwareTrigger

Set software trigger of scope

### Prototype

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

### Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum*           Enter the scope number.

### Description

The `xPCScSoftwareTrigger` function triggers scope *scNum*. The scope must be in the state `Waiting for trigger` for this function to succeed. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Regardless of the trigger mode setting, you can use `xPCScSoftwareTrigger` to force a trigger. In trigger mode `Software`, this function is the only way to trigger the scope.

### See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCIsScFinished`

Methods `SimulinkRealTime.fileScope.trigger`,  
`SimulinkRealTime.hostScope.trigger`, and  
`SimulinkRealTime.targetScope.trigger`

Property `TriggerMode` of `SimulinkRealTime.fileScope`,  
`SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

## xPCScStart

Start data acquisition for scope

### Prototype

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

### Arguments

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

### Description

The `xPCScStart` function starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call `xPCScGetState` to check the state of the scope or, for host scopes that are already started, call `xPCIsScFinished`. Use the `xPCGetScopes` function to get a list of scopes.

### See Also

API functions `xPCGetScopes`, `xPCScGetState`, `xPCScStop`, `xPCIsScFinished`

Scope object method `SimulinkRealTime.fileScope.start`,  
`SimulinkRealTime.hostScope.start`, `SimulinkRealTime.targetScope.start`

# xPCScStop

Stop data acquisition for scope

## Prototype

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

## Arguments

- port*      Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
- scNum*     Enter the scope number.

## Description

The `xPCScStop` function stops the scope *scNum*. This sets the scope to the "Interrupted" state. The scope must be running for this function to succeed. Use `xPCScGetState` to determine the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

## See Also

API functions `xPCGetScopes`, `xPCScStart`, `xPCScGetState`

Scope object methods `SimulinkRealTime.fileScope.stop`, `SimulinkRealTime.hostScope.stop`, `SimulinkRealTime.targetScope.stop`

## xPCSetEcho

Turn message display on or off

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>mode</i>	Valid values are
0	Turns the display off
1	Turns the display on

### Description

On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the real-time application is stopped. When you turn the message display off, the message screen no longer updates. Existing messages remain on the screen as they were.

### See Also

API function `xPCGetEcho`

# xPCSetLastError

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

Change initialization timeout value between development and target computers

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>timeOut</i>	Enter the new communication timeout value.

## Description

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

For example, the function `xPCLoadApp` waits to check whether the model initialization for a new application is complete before returning. When a new real-time application is loaded onto the target computer, the function `xPCLoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCLoadApp` returns a timeout error.

By default, `xPCLoadApp` checks for target readiness for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, models with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. Other functions that communicate with the target computer will wait for *timeOut* seconds before declaring a timeout event.

## See Also

API functions `xPCGetLoadTimeOut`, `xPCLoadApp`, `xPCUnloadApp`



# xPCSetLogMode

Set logging mode and increment value of scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>logging_data</i>	Logging mode and increment value.

## Description

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

## See Also

API function `xPCGetLogMode`

API structure `lgmode`

Property `LogMode` of `SimulinkRealTime.target`

# xPCSetParam

Change value of parameter

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Parameter index.
<i>paramValue</i>	Vector of doubles, assumed to be the size required by the parameter type

## Description

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

## See Also

API functions `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetParam`

# xPCSetSampleTime

Change real-time application sample time

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ts</i>	Sample time for the real-time application.

## Description

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

## See Also

API function `xPCGetSampleTime`

Property `SampleTime` of `SimulinkRealTime.target`

# xPCSetScope

Set properties of scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>state</i>	Enter a structure of type <code>scopedata</code> .

## Description

---

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

---

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

## See Also

API functions `xPCGetScope`, `xPCScGetState`, `scopedata`

## xPCSetStopTime

Change real-time application stop time

### Prototype

```
void xPCSetStopTime(int port, double tfinal);
```

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>tfinal</i>	Enter the stop time, in seconds.

### Description

The `xPCSetStopTime` function sets the stop time of the real-time application to the value in *tfinal*. The real-time application will run for this number of seconds before stopping. Set *tfinal* to `-1.0` to set the stop time to infinity.

### See Also

API function `xPCGetStopTime`

Property `StopTime` of `SimulinkRealTime.target`

# xPCStartApp

Start real-time 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 real-time application loaded on the target computer.

## See Also

API function `xPCStopApp`

Target object method `SimulinkRealTime.target.start`

# xPCStopApp

Stop real-time 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 real-time application loaded on the target computer. The real-time application remains loaded and the parameter changes you made remain intact. If you want to stop and unload an application, use `xPCUnloadApp`.

## See Also

API functions `xPCStartApp`, `xPCUnloadApp`

Target object method `SimulinkRealTime.target.stop`

# xPCTargetPing

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 the program running on your development computer has a fatal error).



## See Also

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

# xPCTgScGetGrid

Get status of grid line for particular scope

## Prototype

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

## Arguments

- port*            Enter the value returned by 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

Get scope mode for displaying signals

## Prototype

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

## Arguments

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

## Return

The `xPCTgScGetMode` function returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

If this function detects an error, it returns `-1`.

## Description

The `xPCTgScGetMode` function gets the mode (`SCMODE_NUMERICAL`, `SCMODE_REDRAW`, `SCMODE_SLIDING`, `SCMODE_ROLLING`) of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCGetScopes` function to get a list of scopes.

## See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `DisplayMode` of `SimulinkRealTime.fileScope`, `SimulinkRealTime.hostScope`, and `SimulinkRealTime.targetScope`

# xPCTgScGetViewMode

Get view mode for target computer display

## Prototype

```
int xPCTgScGetViewMode(int port);
```

## Arguments

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

## Return

The `xPCTgScGetViewMode` function returns the view mode for the target computer screen. If the function detects an error, it returns -1.

## Description

The `xPCTgScGetViewMode` function gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is that of the scope currently displayed on the screen. If the value is 0, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).

## See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `ViewMode` of `SimulinkRealTime.target`

## xPCTgScGetYLimits

Copy *y*-axis limits for scope to array

### Prototype

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

### Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>limits</i>	The first element of the array is the lower limit while the second element is the upper limit.

### Description

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

### See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`

Property `Ylimit` of `SimulinkRealTime.targetScope`

# xPCTgScSetGrid

Set grid mode for scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>grid</i>	Enter a grid value.

## Description

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

## See Also

API functions `xPCGetScopes`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Grid`



# xPCTgScSetMode

Set display mode for scope

## Prototype

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

## Arguments

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Enter the value for the mode.

## Description

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

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

Use the `xPCGetScopes` function to get a list of scopes.

## See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `DisplayMode` of `SimulinkRealTime.targetScope`

# xPCTgScSetViewMode

Set view mode for scope

## Prototype

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

## Arguments

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

## Description

The `xPCTgScSetViewMode` function sets the target computer screen to display one scope with scope number *scNum*. If you set *scNum* to 0, the target computer screen displays all the defined scopes. Use the `xPCGetScopes` function to get a list of scopes.

## See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Property `ViewMode` of `SimulinkRealTime.target`

## xPCTgScSetYLimits

Set *y*-axis limits for scope

### Prototype

```
void xPCTgScSetYLimits(int port, int scNum, const double *Ylimits);
```

### Arguments

- port*        Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.
- scNum*       Enter the scope number.
- Ylimits*     Enter a two-element array.

### Description

The `xPCTgScSetYLimits` function sets the *y*-axis limits for a scope with scope number *scNum* and type `SCTYPE_TARGET` to the values in the double array *Ylimits*. The first element is the lower limit, and the second element is the upper limit. Set both limits to 0.0 to specify autoscaling. Use the `xPCGetScopes` function to get a list of scopes.

### See Also

API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScGetYLimits`

Property `Ylimit` of `SimulinkRealTime.targetScope`

# xPCUnloadApp

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

## See Also

API function `xPCLoadApp`

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



# Configuration Parameters

---

This topic deals with configuration parameters in Simulink Real-Time Explorer and in the MATLAB API.

## Configuration Parameters

### In this section...

- “Simulink Real-Time Options Pane” on page 3-3
- “Automatically download application after building” on page 3-4
- “Download to default target PC” on page 3-5
- “Specify target PC name” on page 3-6
- “Name of Simulink Real-Time object created by build process” on page 3-7
- “Use default communication timeout” on page 3-8
- “Specify the communication timeout in seconds” on page 3-9
- “Execution mode” on page 3-10
- “Real-time interrupt source” on page 3-11
- “I/O board generating the interrupt” on page 3-12
- “PCI slot (-1: autosearch) or ISA base address” on page 3-16
- “Log Task Execution Time” on page 3-17
- “Signal logging data buffer size in doubles” on page 3-18
- “Number of events (each uses 20 bytes)” on page 3-21
- “Double buffer parameter changes” on page 3-22
- “Load a parameter set from a file on the designated target file system” on page 3-23
- “File name” on page 3-24
- “Generate CANape extensions” on page 3-25
- “Enable Stateflow animation” on page 3-26



## Simulink Real-Time Options Pane

Set up general information about building real-time applications, including target, execution, data logging, and other options.

### Configuration

To enable the **Simulink Real-Time Options** pane, you must:

- 1 In the **Code Generation** pane of the Configuration Parameters dialog box, set the **System target file** parameter to `slrt.tlc` or `slrtert.tlc`.

---

**Note:** If you open a model that was originally saved with **System target file** set to `xpctarget.tlc`, the software updates the setting to `slrt.tlc`, and likewise with `xpctargetert.tlc` and `slrtert.tlc`. To retain the updated setting, save the updated model.

---

- 2 Select **C** for the **Language** parameter on the code generation pane.

### Tips

- The default values work for the generation of most real-time applications. If you want to customize the build of your real-time application, set the option parameters to suit your specifications.
- To access configuration 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

“Simulink Real-Time Options Configuration Parameters”

### Automatically download application after building

Enable Simulink Coder to build and download the real-time application to the target computer.

#### Settings

**Default:** on

On

Builds and downloads the real-time application to the target computer.

Off

Builds the real-time 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 Real-Time Application”

## Download to default target PC

Direct Simulink Coder to download the real-time application to the default target computer.

### Settings

**Default:** on



On

Downloads the real-time application to the default target computer. Assumes that you configured a default target computer through Simulink Real-Time Explorer.



Off

Enables the **Specify target PC name** field so that you can enter the target computer to which to download the real-time application.

### Dependency

This parameter enables **Specify target PC name**.

### Command-Line Information

**Parameter:** xPCisDefaultEnv

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

- “Ethernet Link Setup”
- “Serial Link Setup”

### Specify target PC name

Specify a target computer name for your real-time application.

#### Settings

' '

#### Tip

The target computer name appears in Simulink Real-Time Explorer as the target computer node, for example `TargetPC1`.

#### Dependencies

To enable this parameter, set **Download to default target PC**.

#### Command-Line Information

**Parameter:** `xPCTargetPCEnvName`

**Type:** string

**Value:** Any valid target computer

**Default:** ' '

#### See Also

“Simulink Real-Time Explorer Basic Operations”

## Name of Simulink Real-Time 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:** RL320bjectName

**Type:** string

**Value:** 'tg' | valid target object name

**Default:** 'tg'

### See Also

“Real-Time Application Objects”

### Use default communication timeout

Direct Simulink Real-Time software to wait 5 (default) seconds for the real-time application to be downloaded to the target computer.

#### Settings

**Default:** on



Waits the default amount of seconds (5) for the real-time application to be downloaded to the target computer.



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 real-time 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 real-time application to download to the target computer.

### Settings

**Default:** 5

### Tip

Enter the maximum length of time in seconds you want the Simulink Real-Time software to wait for the real-time application to download to the target computer. If the real-time application is not downloaded within this time frame, the software generates an error.

### Dependencies

To enable this parameter, set **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 real-time application execution mode.

#### Settings

**Default:** Real-Time

Real-Time

Executes application as a real-time application.

Freerun

Executes application as fast as possible.

Multirate models cannot be executed in **Freerun** execution mode. On the **Solver** pane in the Configuration Parameters dialog box, set **Tasking mode for periodic sample times** to **SingleTasking**.

#### 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 Simulink Real-Time 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), set the IRQ manually.
- The Simulink Real-Time software treats PCI parallel port plug-in boards like ISA boards. For PCI parallel port plug-in boards, set the IRQ manually.
- Multiple boards can share an 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.

Busmirror EB5100

Specifies that the interrupt source is the Busmirror EB5100 FlexRay™ 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 Fastcom® 422/2-PCI board.

FastComm 422/2-PCI-335

Specifies that the interrupt source is the Fastcom 422/2-PCI-335 board.

FastComm 422/4-PCI-335

Specifies that the interrupt source is the Fastcom 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<sup>®</sup> 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<sup>®</sup> Scramnet+ SC150 board.

**Softing\_CAN-AC2-104**

Specifies that the interrupt source is the Softing<sup>®</sup> CAN-AC2-104 board.

**Softing\_CAN-AC2-PCI**

Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

**Speedgoat\_I0301**

Specifies that the interrupt source is the Speedgoat IO301 FPGA board.

**Speedgoat\_I0302**

Specifies that the interrupt source is the Speedgoat IO302 FPGA board.

**Speedgoat\_I0303**

Specifies that the interrupt source is the Speedgoat IO303 FPGA board.

**Speedgoat\_I0311**

Specifies that the interrupt source is the Speedgoat IO311 FPGA board.

**Speedgoat\_I0312**

Specifies that the interrupt source is the Speedgoat IO312 FPGA board.

**Speedgoat\_I0313**

Specifies that the interrupt source is the Speedgoat IO313 FPGA board.

**Speedgoat\_I0314**

Specifies that the interrupt source is the Speedgoat IO314 FPGA board.

**Speedgoat\_I0321**

Specifies that the interrupt source is the Speedgoat IO321 FPGA board.

**Speedgoat\_I0331**

Specifies that the interrupt source is the Speedgoat IO331 FPGA board.

**UEI\_MF $x$**

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' |  
'Busmirror EB5100' |  
'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\_I0301' |  
'Speedgoat\_I0302' |  
'Speedgoat\_I0303' |  
'Speedgoat\_I0311' |  
'Speedgoat\_I0312' |  
'Speedgoat\_I0313' |  
'Speedgoat\_I0314' |  
'Speedgoat\_I0321' |  
'Speedgoat\_I0331' |  
'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 Simulink Real-Time 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, in the Command Window, type:

```
tg = slrt;  
getPCIInfo(tg, 'installed')
```

#### Command-Line Information

**Parameter:** xPCIOIRQSlot

**Type:** string

**Value:** '-1' | hexadecimal value

**Default:** '-1'

#### See Also

“Simulink Real-Time Options Configuration Parameters”

“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

“Simulink Real-Time Options Configuration Parameters”

“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 cannot exceed the available target computer memory, which the Simulink Real-Time software also uses to hold other items.

#### Tips

- Real-time applications use this buffer to store the time, states, outputs, and task execution time (TET) logs as defined in the Simulink model.
- The maximum value for this option derives from available target computer memory, which the Simulink Real-Time 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 Simulink Real-Time kernel, real-time application, and scopes.

For example, assume that your model has six data items (time, two states, two outputs, and task execution time). If you enter a buffer size of 100000, the target object property `tg.MaxLogSamples` is calculated as `floor(100000 / 6) = 16666`. After the buffer saves 16666 sample points, it wraps and further samples overwrite the older ones.

- Suppose that you enter a logging buffer size larger than the available RAM on the target computer. When you download and initialize the real-time application, the target computer displays a message, **ERROR: allocation of logging memory failed**. To avoid this error, either install more RAM or reduce the buffer size for logging, and then restart the target computer. To calculate the maximum buffer size available for your real-time application logs, divide the amount of available RAM on your target computer by `sizeof(double)`, or 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**

“Simulink Real-Time Options Configuration Parameters”



## 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 or end of an interrupt or iteration of the model. For example, one sample can have four events: the beginning and end of an interrupt, and the beginning and end of an iteration.
- 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

“Execution Profiling for Real-Time Applications”

### Double buffer parameter changes

Use a double buffer for parameter tuning. This setting enables parameter tuning so that the process of changing parameters in the real-time 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 real-time 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, it is queued.
- At the start of execution of the next sample of the real-time task, the queued parameters are updated. This operation increases the task execution time (TET) and can cause a CPU overload error.
- Double buffering leads to a more robust parameter tuning interface, but it increases task execution time and the higher probability of overloads. Under typical conditions, keep double buffering off (default).
- If the real-time application contains model parameters, the software ignores this double buffering setting. Normal parameter tuning occurs.

#### Command-Line Information

**Parameter:** xpcDb1Buff

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

“Simulink Real-Time Options Configuration Parameters”

## 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

“Simulink Real-Time Options Configuration Parameters”

“Save and Reload Parameters Using 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

To enable this parameter, set **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

“Simulink Real-Time Options Configuration Parameters”

## Generate CANape extensions

Enable real-time applications to generate data, such as A2L data, for Vector CANape®.

### Settings

**Default:** off

On

Enables real-time applications to generate data, such as that for A2L, for Vector CANape.

Off

Does not enable real-time 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”

### Enable Stateflow animation

Enables visualization of Stateflow<sup>®</sup> 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 Using Simulink External Mode”



# TLC Options Parameters

---

# TLCOptions Properties

Modify real-time application options

Model options set before code generation to configure the real-time application and the real-time kernel.

To set these options, use the syntax `set_param(model_name, 'TLCOptions', '-aoption_name1=option_value1 -aoption_nameN=option_valueN')`.

Prefix each option name with `-a`. Do not leave spaces around the equals sign. Do not place a comma between consecutive value assignments.

```
set_param(model_name, 'TLCOptions', '-axPCMaxOverloads=20  
-axPCModelStackSizeKB=1024')
```

To read these options, use the syntax `get_param(model_name, 'TLCOptions')`.

```
get_param(model_name, 'TLCOptions')
```

```
ans =
```

```
-axPCMaxOverloads=20 -axPCModelStackSizeKB=1024
```

To remove these options, use the syntax `set_param(model_name, 'TLCOptions', '')`.

```
set_param(model_name, 'TLCOptions', '')
```

## Target Computer Overload

**xPCMaxOverloads** — Number of acceptable target computer overloads

0 (default) | scalar

When `xPCMaxOverloads` is set to a value, such as 3, the Simulink Real-Time software will stop execution with a CPU overload at the following overload (the fourth).

Example: `-axPCMaxOverloads=3`

**xPCMaxOverloadLen** — Number of contiguous acceptable overloads

0 (default) | scalar

You must specify a value that is the same or less than the value for `xPCMaxOverloads`.

When `xPCMaxOverloadLen` is set to a value, such as 2, the software will stop execution with a CPU overload at the following contiguous overload (the third).

Example: `-axPCMaxOverloadLen=2`

### **xPCStartupFlag** — Number of executions of the model at startup

1 (default) | scalar

Causes the software to temporarily disable the timer interrupt during model execution. After the model finishes the first `xPCStartupFlag` number of executions, the software reenables the timer interrupt, which invokes the next execution for the model.

Example: `-axPCStartupFlag=3`

## Target Computer Memory

### **xPCModelStackSizeKB** — Size of stack memory on the target computer, in kilobytes

512 (default) | scalar

Sets the number of kilobytes of stack memory that are allocated to real-time threads on the target computer

Example: `-axPCModelStackSizeKB=1024`

## Polling Mode

### **xpcCPUClockPoll** — Target computer CPU clock rate, in MHz

0 (default) | scalar

Switches the kernel from interrupt mode to polling mode. When **Execution mode** is **Real-Time**, a nonzero value causes the real-time application to perform a busy wait at the specified polling rate, assumed to be the target computer CPU clock rate. If the value is 0 or if the option is not defined, the kernel executes in interrupt mode.

Example: `-axpcCPUClockPoll=1200`

## More About

- “Maximizing Target Computer CPU Usage”

- “Polling Mode”

# Signal and Parameter Instruments

---

- “Instruments and Instrument Panels” on page 5-2
- “Create Instrument Panel” on page 5-12
- “Configure Instrument for Parameter Tuning” on page 5-13
- “Configure Instruments for Signal Display” on page 5-16
- “Save and Load Instrument Panels” on page 5-20
- “Save and Restore Layouts” on page 5-21
- “Run Instrumented Model” on page 5-22
- “Prepare Deployment Environment for Export” on page 5-25
- “Create Parameter Groups for Export” on page 5-27
- “Create Signal Groups for Export” on page 5-29
- “Prepare Instrument Panel Configuration for Export” on page 5-31
- “Export Instrument Panel Configuration” on page 5-33
- “Deploy and Run Standalone Configuration” on page 5-35
- “Instruments — Alphabetical List” on page 5-37

# Instruments and Instrument Panels

In this section...
“Instrument Selection” on page 5-2
“Simulink Real-Time Explorer Within MATLAB and as Standalone Executable” on page 5-6
“Guidelines for Exporting Instrumented Models” on page 5-9

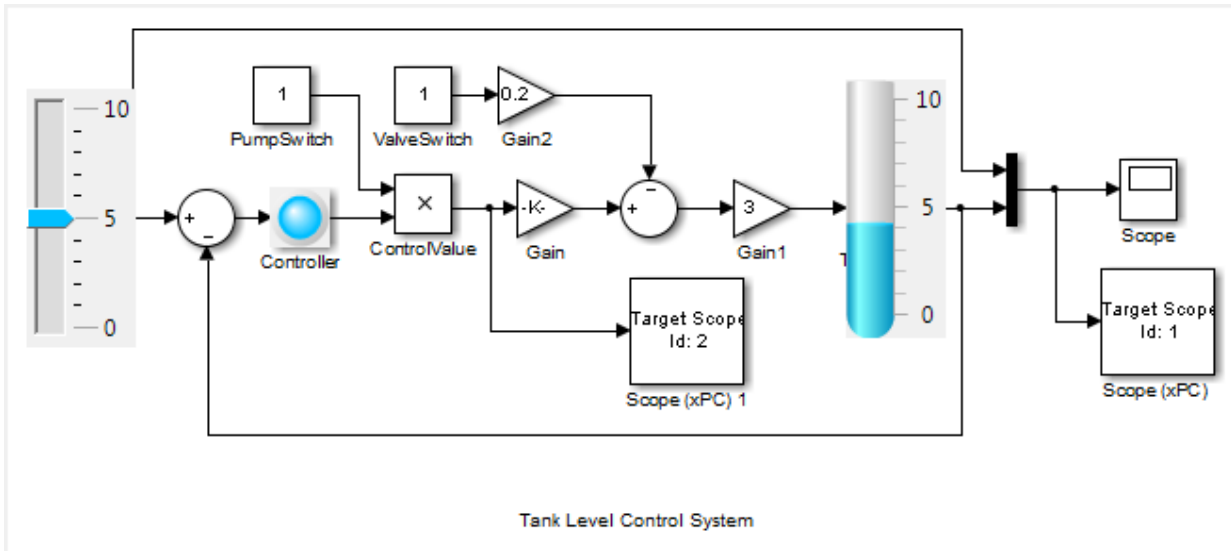
You can use Simulink Real-Time Explorer instruments and instrument panels to control a real-time system model running on a target computer.

For example, suppose that you develop a model of a cruise control running on a target computer. To test the model, you connect the model to automobile drive train hardware. You can then develop and export an instrument panel to a deployment laptop. Use that panel to start and stop the cruise control model.

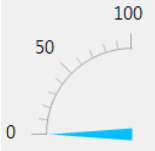
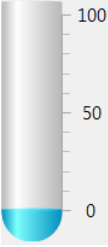
Later in the test process, you deploy a drive-train (plant) model to a standalone target computer. You can then develop and export an instrument panel to a deployment laptop. Use that panel to test the actual cruise control hardware.

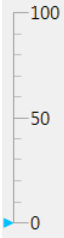



## Instrument Selection

To instrument a model, populate an instrument panel with instruments compatible with the parameters and signals that they represent in the model. You can add layout elements, such as a background diagram, to make the relationship between the instruments and the model clearer.

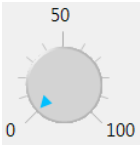


The table shows some uses for signal display and parameter control instruments and for layout elements

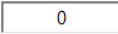

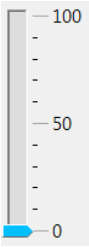

To	Requiring	Use
<ul style="list-style-type: none"> <li>Show the pressure in a container.</li> <li>Show the speed of a vehicle.</li> <li>Show current or voltage in a circuit.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical instruments that show approximate value by angular displacement</li> </ul>	 <p>GaugeAngular</p>
<ul style="list-style-type: none"> <li>Show the level of fluid in a container.</li> <li>Show the pressure in a pipe.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical instruments that show approximate value by vertical displacement</li> </ul>	 <p>GaugeFluidLevel</p>

To	Requiring	Use
<ul style="list-style-type: none"> <li>Show the pressure in a container.</li> <li>Show audio output power.</li> <li>Show current or voltage in a circuit.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical instruments that show approximate value by linear displacement</li> </ul>	 <p>GaugeLinear</p>
<ul style="list-style-type: none"> <li>Show traffic on a digital bus.</li> <li>Show the state of a state machine.</li> </ul>	<ul style="list-style-type: none"> <li>Integer-valued data</li> <li>Analogy to physical instruments that show hexadecimal values</li> </ul>	 <p>HexadecimalDisplay</p>
<ul style="list-style-type: none"> <li>Show on-off state of a switch.</li> <li>Show on-off state of a bidirectional pin.</li> </ul>	<ul style="list-style-type: none"> <li>Boolean data</li> <li>Analogy to physical instruments that show value by lights turning on and off</li> </ul>	 <p>LED</p>
<ul style="list-style-type: none"> <li>Show a temperature measurement to given precision.</li> <li>Show a voltage measurement to given precision.</li> <li>Show a date and time</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical instruments that show real values in decimal and other format.</li> </ul>	 <p>NumericDisplay</p>

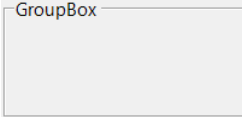
**Parameter Tuning Instruments**

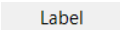
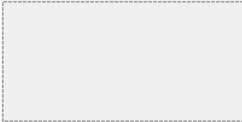
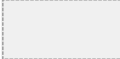
To	Requiring	Use
<ul style="list-style-type: none"> <li>Control gain of a radio receiver.</li> <li>Control amplification of a radio transmitter.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical controls that set approximate value by angular displacement</li> </ul>	 <p>Knob</p>



To	Requiring	Use
<ul style="list-style-type: none"> <li>Enter initial set point control value for thermostat.</li> <li>Enter seed value for random number generator.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical controls that set exact numeric value</li> </ul>	 NumericEntry
<ul style="list-style-type: none"> <li>Control car radio audio volume using step-increment.</li> <li>Smoke test controller range in small number of clicks.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical controls that set initial value and step increment</li> </ul>	 NumericUpDownEntry
<ul style="list-style-type: none"> <li>Control frequency of a radio receiver.</li> <li>Control pressure valve setting.</li> </ul>	<ul style="list-style-type: none"> <li>Real-valued data</li> <li>Analogy to physical controls that set approximate values by linear displacement</li> </ul>	 Slider
<ul style="list-style-type: none"> <li>Turn on a power supply.</li> <li>Close a gate valve.</li> </ul>	<ul style="list-style-type: none"> <li>Boolean data</li> <li>Analogy to physical controls that are either on or off</li> </ul>	 SwitchLED

### Layout Elements

To	Requiring	Use
<ul style="list-style-type: none"> <li>Group electrical instruments with label Power.</li> <li>Group pressure instruments with label Pressure.</li> </ul>	<ul style="list-style-type: none"> <li>Design-time box resizing</li> <li>Run-time static display</li> </ul>	 GroupBox

To	Requiring	Use
<ul style="list-style-type: none"> <li>Label the <b>Current</b> and <b>Voltage</b> instruments.</li> <li>Label the <b>Temperature</b> and <b>Pressure</b> instruments.</li> </ul>	<ul style="list-style-type: none"> <li>Design-time box resizing, left-right and up-down text alignment</li> <li>Run-time static display</li> </ul>	 Label Label
<ul style="list-style-type: none"> <li>Group the <b>Power</b> group and the <b>Pressure</b> group and scroll between them.</li> <li>Group a picture box and instruments together to display the instruments upon a circuit diagram.</li> </ul>	<ul style="list-style-type: none"> <li>Design-time box resizing</li> <li>Run-time box scrolling</li> </ul>	 Panel
<ul style="list-style-type: none"> <li>Insert a circuit diagram on panel behind current, voltage, and power instruments.</li> <li>Insert a circulation diagram on panel behind pressure and temperature instruments.</li> </ul>	<ul style="list-style-type: none"> <li>Design-time image stretch, zoom, center, and autosize</li> <li>Run-time static display</li> </ul>	 PictureBox

## Simulink Real-Time Explorer Within MATLAB and as Standalone Executable

You can use instrument panels from Simulink Real-Time Explorer in two modes:

- Within MATLAB running on a development computer.
- As a standalone executable running on a deployment computer.

When you run Simulink Real-Time Explorer from within MATLAB, you can work with the full capabilities of Simulink Real-Time Explorer.

Capability	MATLAB and Standalone Executable	MATLAB Only
Environment configuration	<ul style="list-style-type: none"> <li>View communication parameters.</li> </ul>	<ul style="list-style-type: none"> <li>Configure communication parameters.</li> </ul>

Capability	MATLAB and Standalone Executable	MATLAB Only
Control	<ul style="list-style-type: none"> <li>• Browse target computer file system.</li> </ul>	<ul style="list-style-type: none"> <li>• Configure target computer settings</li> <li>• Configure target computer startup</li> </ul>
	<ul style="list-style-type: none"> <li>• Run real-time applications on the target computer.</li> <li>• Connect to and disconnect from the target computer.</li> <li>• Change stop time and sample times without regenerating code.</li> <li>• Record task execution time during or after last run.</li> </ul>	<ul style="list-style-type: none"> <li>• Load and unload real-time applications.</li> </ul>
	<ul style="list-style-type: none"> <li>• Start and stop instrument panels.</li> <li>• Use instrument panels to interact with application.</li> </ul>	<ul style="list-style-type: none"> <li>• Create graphical instrument panels for acquiring signals and tuning parameters.</li> <li>• Save and load instrument panels.</li> </ul>
	<ul style="list-style-type: none"> <li>• From signal groups, monitor signals.</li> <li>• Add and configure host, target, or file scopes.</li> <li>• From signal groups, attach signals to or remove signals from scopes.</li> <li>• From signal groups, attach signals to instruments.</li> <li>• Start and stop scopes.</li> </ul>	<ul style="list-style-type: none"> <li>• From model hierarchy, monitor signals.</li> <li>• From model hierarchy, add signals to or remove signals from scopes.</li> <li>• From model hierarchy, attach signals to instruments.</li> <li>• Create, save, and load signal groups.</li> </ul>
Signal acquisition		

Capability	MATLAB and Standalone Executable	MATLAB Only
Parameter tuning	<ul style="list-style-type: none"> <li>From parameter groups, display and tune parameters.</li> <li>From parameter groups, attach parameters to instruments.</li> </ul>	<ul style="list-style-type: none"> <li>From model hierarchy, display and tune parameters.</li> <li>From model hierarchy, attach parameters to instruments.</li> <li>Create, save, and load parameter groups.</li> </ul>
Window configuration	<ul style="list-style-type: none"> <li>Make multiple workspaces visible simultaneously.</li> <li>Move workspaces around the window.</li> </ul>	<ul style="list-style-type: none"> <li>Export model configuration as a standalone executable.</li> <li>Save and restore model configuration layouts.</li> </ul>

When you run Simulink Real-Time Explorer as a standalone executable, it has the following restrictions:

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.
- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. If you rename the target computer, update the **TargetName** parameter for each instrument to maintain the connection to the real-time application.
- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.
- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.

You cannot move a signal from one signal group to another or create or load a new signal group.

- You can access only parameters in parameter groups that you loaded before you exported the configuration.

You cannot move a parameter from one parameter group to another or create or load a new parameter group.

- You cannot save session layouts. If you close a window, you can restore the original layout using **File > Restore Original View**.

## Guidelines for Exporting Instrumented Models

- “Deployment Environment” on page 5-9
- “Signal Groups” on page 5-10
- “Parameter Groups” on page 5-10
- “Instrument Panels” on page 5-10
- “Window Layout” on page 5-11

Before exporting an instrumented model, set up the environment, groups, panels, and window layout the way you want them to be in the standalone executable. These features cannot be changed after you export the configuration. Follow these guidelines:

### Deployment Environment

For each deployment computer:

- Verify that the deployment computer meets the requirements for running the standalone Simulink Real-Time Explorer executable.

Record whether the deployment computer has a 64-bit or a 32-bit architecture. You need this information to run the export program.

- Configure the deployment computer to have the same drive letter and folder paths as the drive and paths on the development computer. For example, if the instrument panel, signal group, and parameter group files are in `C:\workdir`, the deployment computer must also have `C:\workdir`.

For each target computer:

- Verify that the target computer meets the requirements for running the real-time application.
- In Simulink Real-Time Explorer, configure the target and communication settings required to connect each deployment computer to each target computer on which a real-time application runs.

You can have only one target computer node for each unique **IP address** setting.

- Configure the **Boot mode** setting for each target computer as **Stand Alone**.
- Optionally, rename the target computer session from **TargetPCx** to something more specific to your application.
- As a test, build and download a real-time application to each target computer connected to the development computer running Simulink Real-Time Explorer.

The real-time application loaded on the target computer is the same application that you intend to access with the standalone executable.

### Signal Groups

- To access signals, add them from the model hierarchy to a signal group.
- To include a signal group in the standalone package, load it into the current session.
- From the deployed standalone Simulink Real-Time Explorer package, you can replace one signal-to-instrument binding with another. The new signal and instrument must both be present in the exported executable. The range of the instrument must be compatible with the range of the new signal.

### Parameter Groups

- To access parameters, add them from the model hierarchy to a parameter group.
- To include a parameter in the standalone package, load it into the current session.
- You can replace one parameter-to-instrument binding with another, provided the new parameter and instrument are both present in the exported executable. The range of the instrument must be compatible with the range of the new parameter.

### Instrument Panels

- To interact with multiple target computers, create a separate instrument panel for each separate real-time application/target computer pair.
- To include an instrument panel in the standalone package, load it into the current Simulink Real-Time Explorer session.
- If you renamed the target computer, update the **TargetName** parameter for each instrument to maintain the connection to the real-time application.

## Window Layout

- You can configure which windows the software opens on startup by opening the windows and arranging them accordingly.

When you export the model configuration, the software includes the windows layout in the standalone package.

- To return to the initial standalone executable layout, click **File > Restore Initial View**.


## Related Examples

- “Standalone Boot Method”
- “Simulink Real-Time Explorer Run Within MATLAB”
- “Simulink Real-Time Explorer Deployed as Standalone Executable”

### Create Instrument Panel

In this step, you create and save an instrument panel for the `xpctank` model. Start by building and downloading the real-time application to the target computer, running Simulink Real-Time Explorer, and connecting Explorer to the target computer.

To create an instrument panel:

- 1 In the **Panels** pane, right-click the **Instrument Panels** node, and then click **Add New**.
- 2 Type a name and folder in the **Name** and **Location** text boxes. Give the panel a name like `xpctank_instr_design.slrtip`.
- 3 To save your instrument panel, click in the instrument panel window, and then click the Save icon .

The next task is “Configure Instrument for Parameter Tuning”.

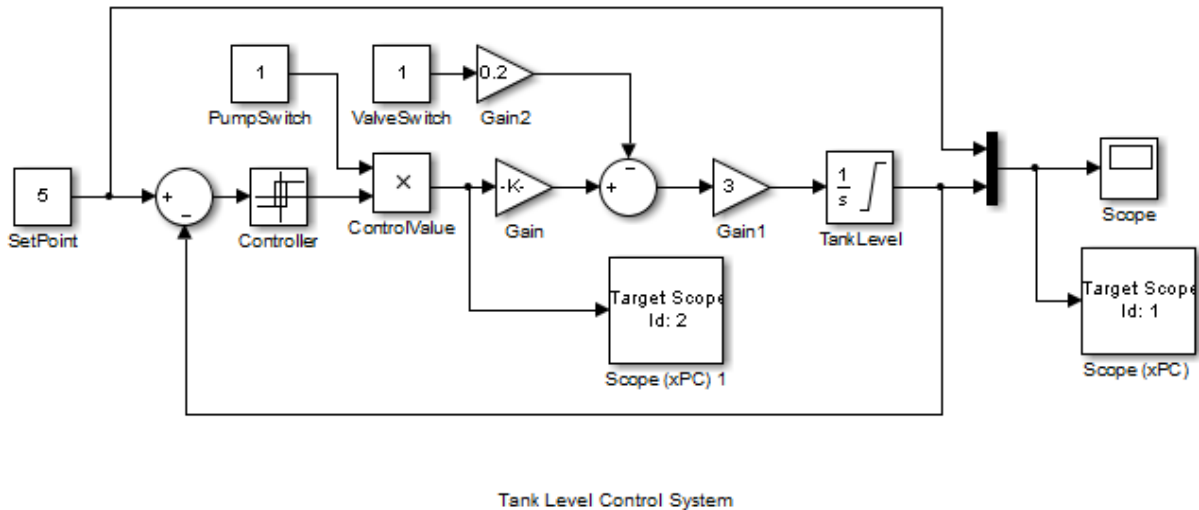
#### Related Examples

- “Save and Restore Layouts”



## Configure Instrument for Parameter Tuning

In this step, you select and configure an instrument to tune a parameter in the xpctank model. You must have previously created the xpctank\_instr\_design.slrtip instrument panel (see “Create Instrument Panel”).



The parameter characteristics are listed in this table.

Name	Type	Range	Purpose
SetPoint	Numeric	0–10 units	Represents the level at which the controller maintains the tank fluid level. You do not have to set it to an exact value.

The Slider instrument meets the requirement for SetPoint. To set an exact numeric value, use, for example, a NumericEntry instrument.

To select and configure the instrument:



- 1 Load the instrument panel.

In the **Panels** pane, right-click the **Instrument Panels** node and select **Existing**. From the list, select xpctank\_instr\_design.slrtip.

- 2 Select the instrument.


From the **Palette** pane, drag a Slider instrument into the `xpctank_instr_design.slrtip` instrument panel.

- 3 Bind the parameter to the instrument.

To bind the `SetPoint` parameter to the Slider instrument, open the Parameter workspace for model `xpctank` ( on the toolbar). Select and drag the Parameter icon  next to parameter `SetPoint` to the Slider instrument.

A small copy of the Parameter icon appears next to the Slider instrument.

- 4 Set the instrument range.

Select the Slider instrument, and then click the Tasks icon  in the top right corner.

- 5 In the **Slider Tasks** dialog box, set property **Min** to **0** and property **Span** to **10**.

- 6 Select and configure a label.

From the **Palette** pane, drag a Label layout item to under the Slider instrument.

- 7 Click the Label element.

- 8 In the **Properties** pane, scroll down to the **Appearance** node. Set the **Text** property to `Set Point`.

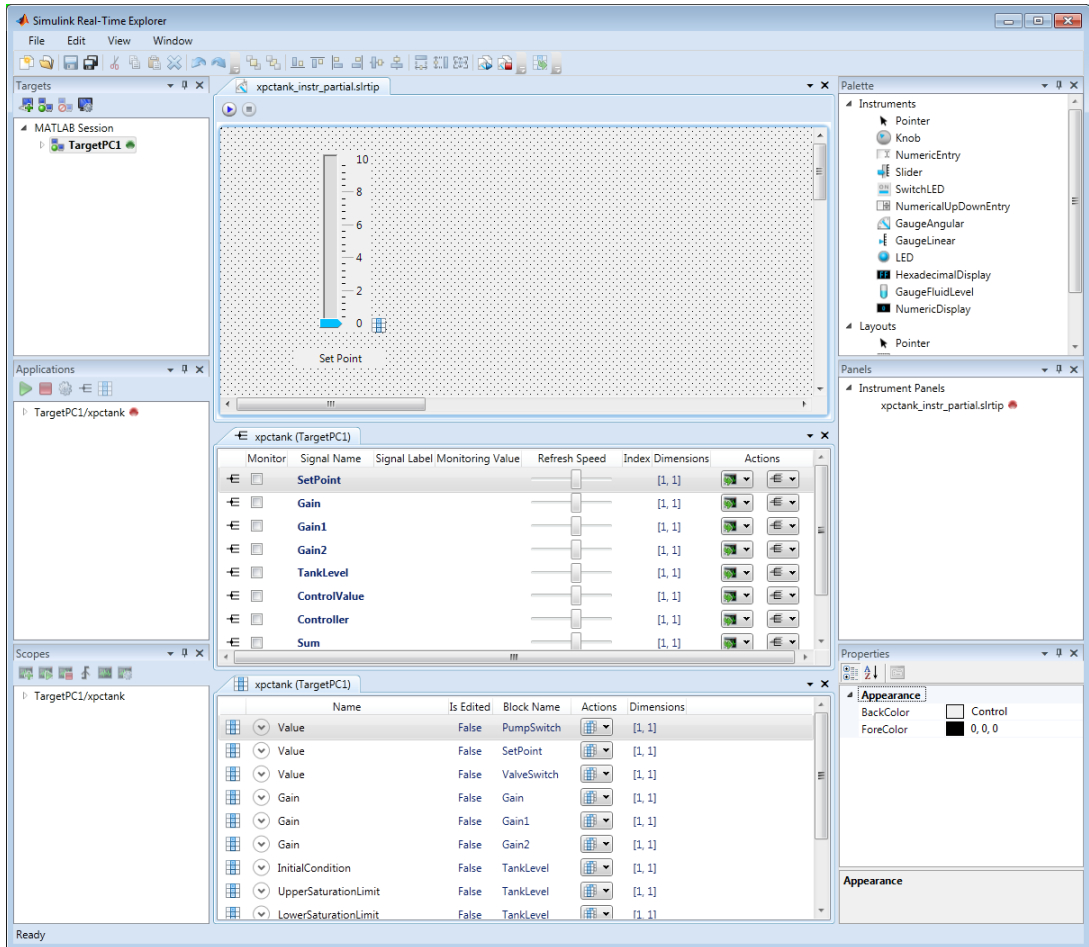
- 9 Scroll down to the **TextAlign** property. Click the down arrow and select the center of the nine blocks presented.

The **TextAlign** property becomes `MiddleCenter`.

- 10 Save the instrument panel.

Click in the instrument panel window, and then click the Save icon .

At the end of this task, the Simulink Real-Time Explorer window looks like this figure.



The next task is “Configure Instruments for Signal Display”.

## Related Examples

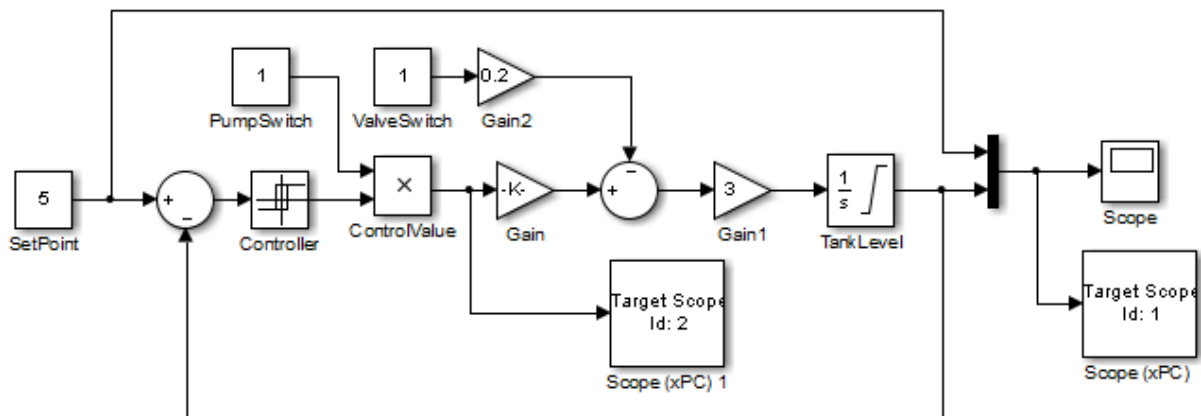
- “Save and Restore Layouts”

## More About

- “Instruments and Instrument Panels”

## Configure Instruments for Signal Display

In this step, you select and configure instruments to display two signals in the xpctank model. You must have previously created the xpctank\_instr\_design.slrtip instrument panel (see “Create Instrument Panel”).



Tank Level Control System

The signal characteristics are listed in this table.

Name	Type	Range	Purpose
TankLevel	Numeric	0–10 units	Represents the current tank fluid level. You do not have to display an exact value.
ControlValue	Boolean	1, 0	Represents the state of the pump (on or off).

- The GaugeFluidLevel instrument meets the requirements for TankLevel. To display an exact numeric value, use, for example, a NumericDisplay instrument.
- The LED instrument meets the requirements for ControlValue.

To select and configure each instrument:

- 1 Load the instrument panel.

In the **Panels** pane, right-click the **Instrument Panels** node and select **Existing**. From the list, select `xpctank_instr_design.slrtip`.

- 2 Select the instrument.

From the **Palette** pane, drag a `GaugeFluidLevel` instrument into the `xpctank_instr_design.slrtip` instrument panel.

- 3 Bind the signal to the instrument.

To bind the `TankLevel` signal to the `GaugeFluidLevel` instrument, open the Signal workspace for model `xpctank` (☰ on the toolbar). Select and drag the Signal icon ☰ next to signal `TankLevel` to the `GaugeFluidLevel` instrument.

A small copy of the Signal icon appears next to the `GaugeFluidLevel` instrument.

- 4 Set the instrument range as required.

Select the `GaugeFluidLevel` instrument, and then click the Tasks icon (☰) in the top right corner.

- 5 In the **GaugeFluidLevel Tasks** dialog box, set property **Min** to 0 and property **Span** to 10.

- 6 Select and configure a label.

From the **Palette** pane, drag a Label layout item to under the `GaugeFluidLevel` instrument.

- 7 Click the Label element.

- 8 In the **Properties** pane, scroll down to the **Appearance** node. Set the **Text** property to `Tank Level`.

- 9 Scroll down to the **TextAlign** property. Click the down arrow and select the center of the nine blocks presented.

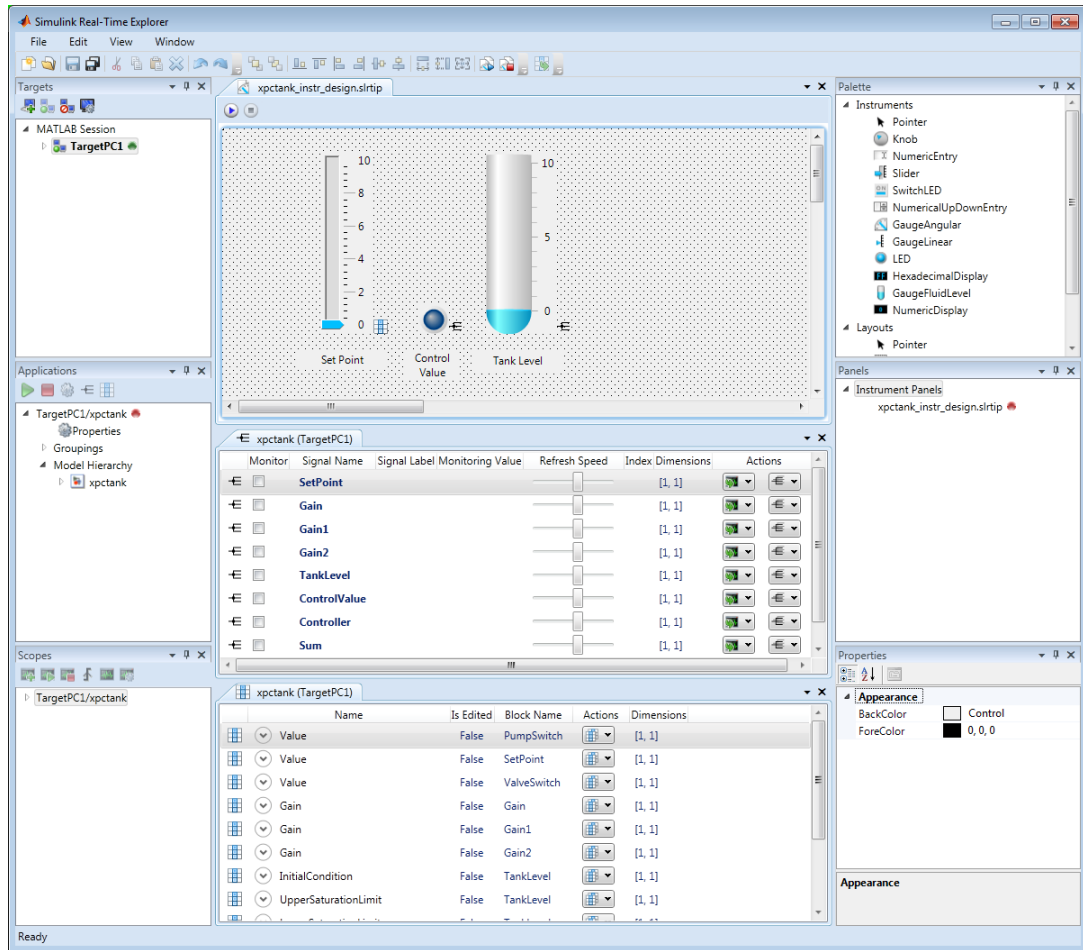
The **TextAlign** property becomes `MiddleCenter`.

- 10 Save the instrument panel.

Click in the instrument panel window, and then click the Save icon .

Using a similar procedure, add an LED instrument to the instrument panel and bind signal `ControlValue` to it. Label the LED `Control Value`.

At the end of this task, the Simulink Real-Time Explorer window looks like this figure.



The next task is “Run Instrumented Model”.

### Related Examples


- “Save and Restore Layouts”

## **More About**

- “Instruments and Instrument Panels”

### Save and Load Instrument Panels

As you are developing instrument panels to control your model, you can save your panels or load existing panels. You can load more than one instrument panel at a time.

- To save your instrument panel, click in the **Panels** pane, and then click the Save icon .
- To load an existing instrument panel, in the **Panels** pane, right-click the **Instrument Panels** node and select **Existing**.



## Save and Restore Layouts

As you are configuring Simulink Real-Time session layouts, you can save your layout or restore a previous layout. Saving a layout preserves the following information:

- The position of the open panes and tabs.
- The target computer connections.
- The instrument panels that you loaded.
- The signal and parameter groups that you loaded.

Saving a layout saves the **Scopes** pane position, but it does not save the state of the scopes in the pane. In particular, if you add a scope within a Simulink Real-Time session, the software does not restore the new scope with the rest of the layout.




- To save a session layout, click **File > Save Layout**.
- To restore a session layout, click **File > Restore Layout**.

# Run Instrumented Model

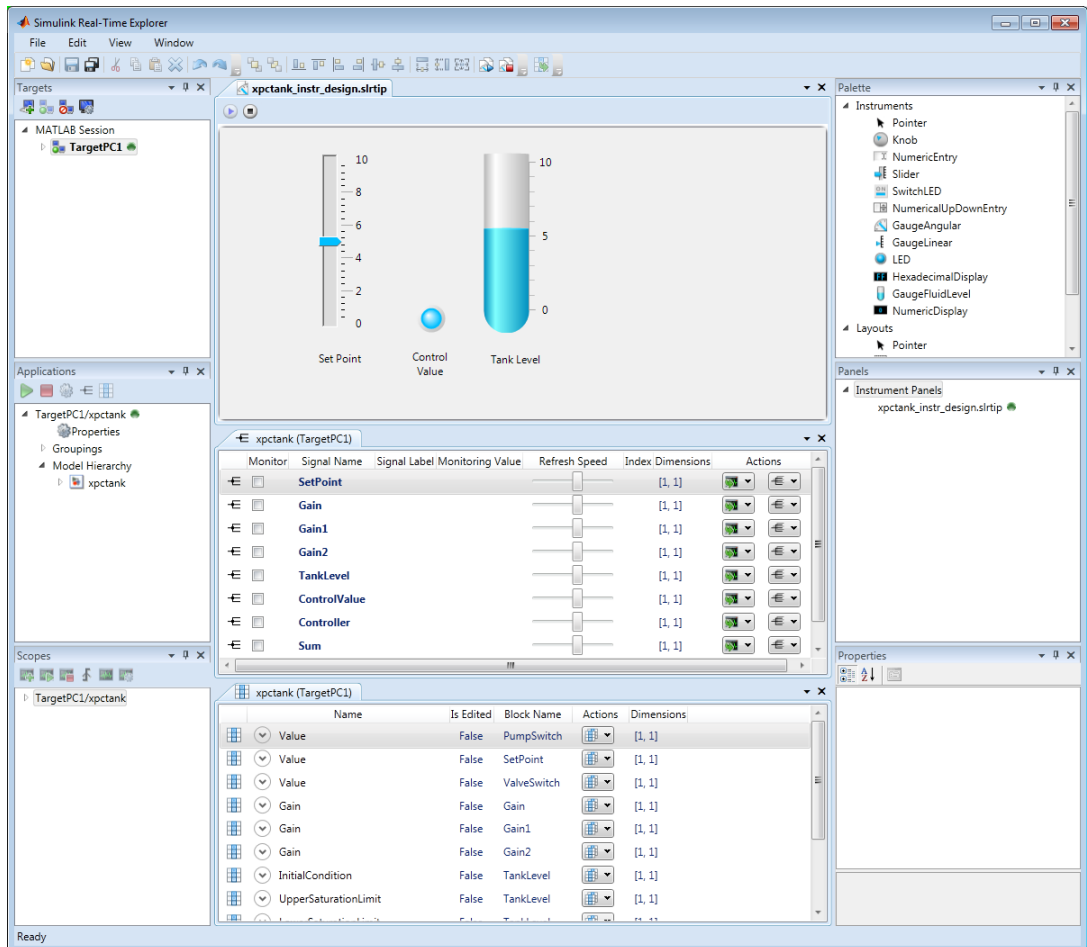
This example shows how to run the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Configure Instrument for Parameter Tuning” and “Configure Instruments for Signal Display”.



- 1 Load the instrument panel.

In the **Panels** pane, right-click the **Instrument Panels** node and select **Existing**. From the list, select `xpctank_instr_design.slrtip`.

- 2 Set property **Stop time** to `inf` in the **Applications** pane ( on the toolbar).
- 3 To start the instrument, in the `xpctank_instr_design.slrtip` instrument panel, click the Run Instrument icon .
- 4 To start execution, in the **Applications** pane, click the real-time application, and then click the Start icon  on the toolbar.
- 5 Using the Slider instrument, set the tank level to the required value, such as 5.

The tank level rises to and oscillates around the set point, as shown in this figure.



- 6 To stop execution, in the **Applications** pane, click the real-time application, and then click the Stop icon .
- 7 To stop the instruments, in the `xpctank_instr_design.slrtip` instrument panel, click the Stop Instrument icon .

## Related Examples

- “Simulink Real-Time Explorer Deployed as Standalone Executable”

## **More About**

- “Instruments and Instrument Panels”

## Prepare Deployment Environment for Export

In this step, verify that each combination of deployment computer and target computer works together. Each target computer must run in standalone mode.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Simulink Real-Time Explorer Run Within MATLAB”.

For each deployment computer, do the following:

- 1 Verify that the CPU and operating system meet the requirements for running the executable interface.  
  
Record whether the deployment computer has a 64-bit or a 32-bit architecture. You need this information to run the export program.
- 2 Configure the deployment computer to have the same drive letter and folder paths as the development computer. For example, if the instrument panel, signal group, and parameter group files are in `C:\workdir`, the deployment computer must also have the drive and folder `C:\workdir`.

For each target computer, do the following:

- 1 Verify that the target computer CPU and operating system meet the requirements for running the Simulink Real-Time kernel.
- 2 Verify that the Simulink Real-Time Explorer **Targets** pane contains a target computer node representing each target computer that you intend to access from the deployment computer.

If you rename a target computer node, make the corresponding change in the **Bindings > TargetName** property for each instrument.

- 3 Verify that the settings in the **Host-to-Target communication** tab match the requirements of the target computer.

You can have only one target computer node for each unique **IP address** setting.

- 4 Verify that the settings in the **Target settings** tab match the capabilities of the target computer.
- 5 Prepare and copy the required kernel and real-time application files to the target computer. Then, in the **Boot configuration** tab, set **Boot mode** to **Stand Alone**.

- 6 Connect the target computer to the deployment computer and restart the target computer. Verify that the target computer loads the Simulink Real-Time kernel and starts the real-time application.

The next task is “Create Parameter Groups for Export”.

### **Related Examples**

- “Development Computer Hardware”
- “Computer Hardware”
- “Ethernet Link Setup”
- “Target Computer Settings”
- “Standalone Boot Method”

### **More About**

- “Instruments and Instrument Panels”




## Create Parameter Groups for Export

Create parameter groups to focus the display on key components and subsystems.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Prepare Deployment Environment for Export”.

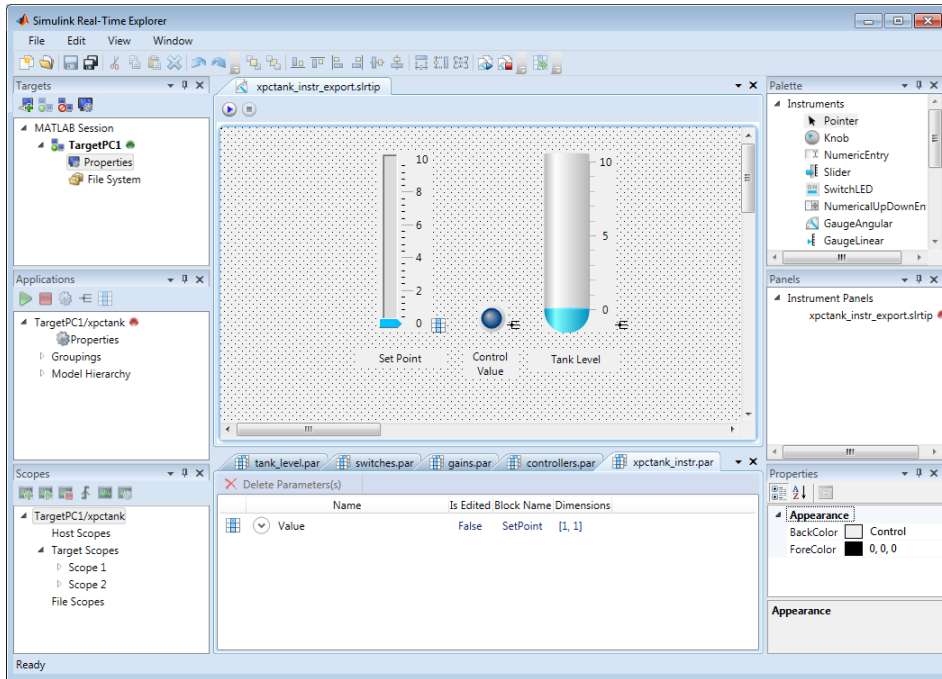
- 1 In the Application window, for your application, right-click the node **Groupings** and select **New Parameter Group**.

To open an existing parameter group, in the Application window, right-click the **Groupings** node for your application. Click **Existing Group** and select the parameter group that you want.

- 2 In the dialog box, give the group a name, such as `xpctank_instr.par`. Leave the path set to the current folder.
- 3 Open the Parameter workspace for model `xpctank` ( on the toolbar). Select and drag the Parameter icon  next to parameter `SetPoint` into the window for the new parameter group.
- 4 In the Application window, click in your application, and then click the Save icon  to save your parameter group.

Using the same procedure, create parameter groups for the blocks representing tank level, switches, gains, and controllers. The corresponding parameter groups are `tank_level.par`, `switches.par`, `gains.par`, and `controllers.par`.

The `xpctank_instr_design.slrtip` configuration looks like the figure.



The next task is “Create Signal Groups for Export”.

### Related Examples

- “Create Parameter Groups Using Simulink Real-Time Explorer”

### More About

- “Instruments and Instrument Panels”




## Create Signal Groups for Export

Create signal groups to focus the display on key components and subsystems.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Prepare Deployment Environment for Export”.

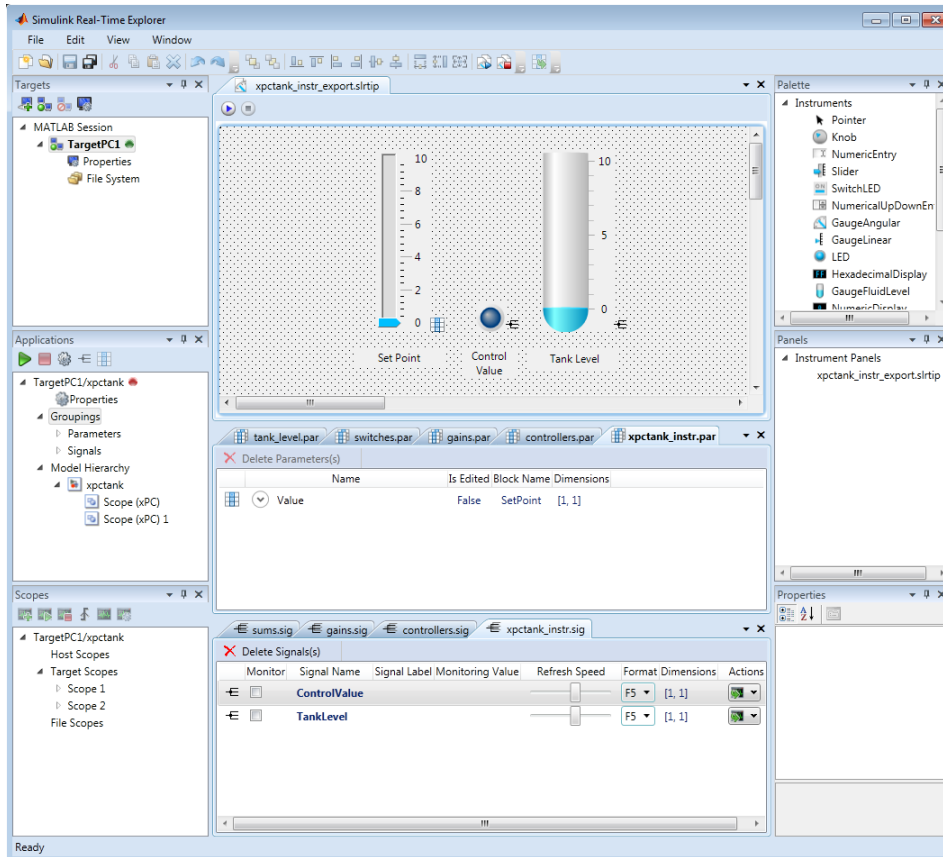
- 1 In the Application window, for your application, right-click the node **Groupings** and select **New Signal Group**.

To open an existing signal group, in the Application window, right-click the **Groupings** node for your application. Click **Existing Group** and select the signal group that you want.

- 2 In the dialog box, give the group a name, such as `xpctank_instr.sig`. Leave the path set to the current folder.
- 3 Open the Signal workspace for model `xpctank` (☰ on the toolbar). Select and drag the Signal icon ☰ next to signals `ControlValue` and `TankLevel` into the window for the new signal group.
- 4 In the Application window, click in your application, and then click the Save icon  to save your signal group.

Using the same procedure, create signal groups for the blocks representing sums, gains, and controllers. The corresponding signal groups are `sums.sig`, `gains.sig`, and `controllers.sig`.

The `xpctank_instr_design.slrtip` configuration looks like the figure.



The next task is “Prepare Instrument Panel Configuration for Export”.

### Related Examples

- “Create Signal Groups Using Simulink Real-Time Explorer”

### More About

- “Instruments and Instrument Panels”

## Prepare Instrument Panel Configuration for Export

In this step, load the instrument panels for the instrumented model and resize and lay out the Simulink Real-Time Explorer windows.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Create Signal Groups for Export” and “Create Parameter Groups for Export”.

---

**Note:** When you run the standalone executable, you cannot access the model hierarchy. You can access only instrument panels and windows that were open when you exported the configuration. You can access only signals and parameters that were in loaded signal and parameter groups when you exported the configuration.

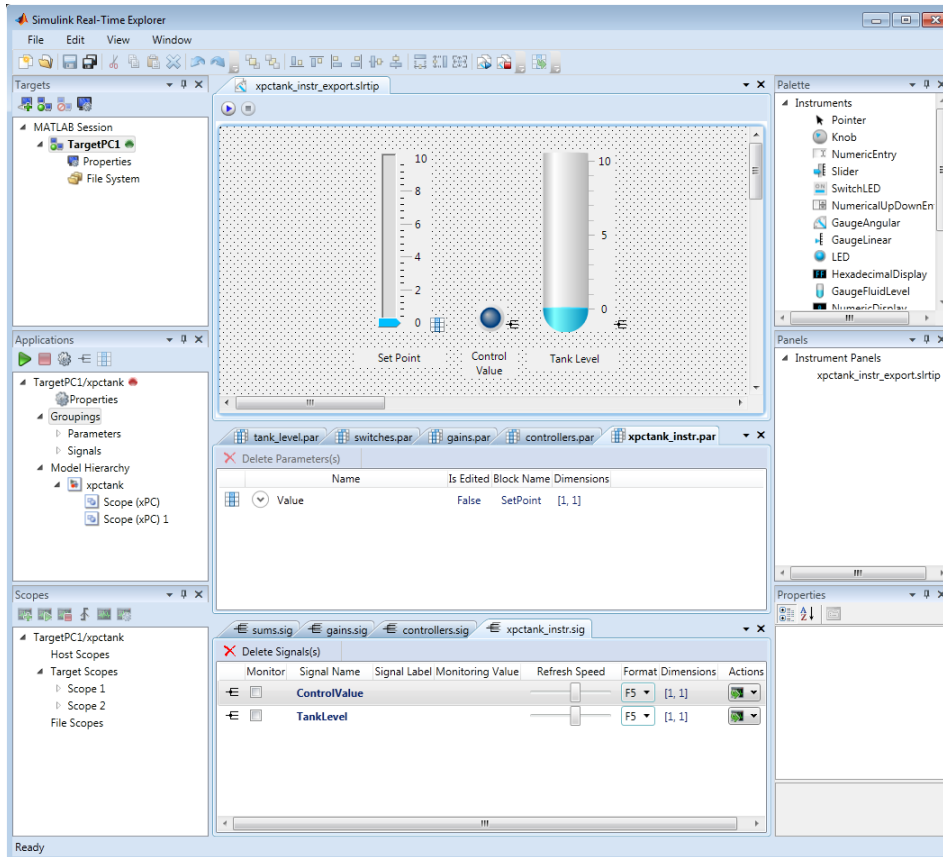
---

- 1 Load your instrument panels into Simulink Real-Time Explorer.

Here, the panel is `xpctank_instr_design.slrtip`.

- 2 Open, lay out, and resize the windows that you want the standalone executable to open.
- 3 Save each instrument panel.

The `xpctank_instr_design.slrtip` configuration looks like the figure.



The next task is “Export Instrument Panel Configuration”.

### More About

- “Instruments and Instrument Panels”

## Export Instrument Panel Configuration

In this step, export the Simulink Real-Time Explorer instrument panel configuration as a standalone executable.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Prepare Instrument Panel Configuration for Export”.

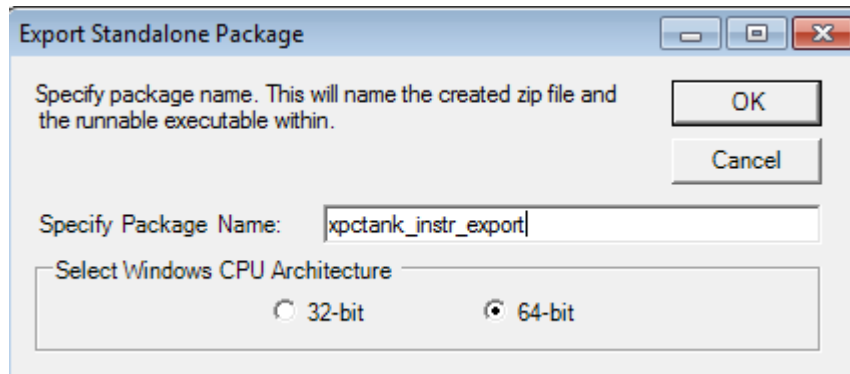
---

**Note:** When you run the standalone executable, you cannot access the model hierarchy. You can access only instrument panels and windows that were open when you exported the configuration. You can access only signals and parameters that were in loaded signal and parameter groups when you exported the configuration.

---

- 1 To export the configuration as a standalone executable, click **File > Export**.
- 2 In the **Specify Package Name** text box, type `xpctank_instr_export`.
- 3 For Windows CPU architecture, select **32-bit** or **64-bit**.

The **Export Standalone Package** dialog box looks like the figure.



- 4 Click **OK**.

The software generates a file named `xpctank_instr_export.zip` in the current folder.

The next task is “Deploy and Run Standalone Configuration”.

## **More About**

- “Instruments and Instrument Panels”

## Deploy and Run Standalone Configuration

In this step, deploy the standalone executable to the deployment computer.

The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in “Export Instrument Panel Configuration”.

---

**Note:** When you run the standalone executable, you cannot access the model hierarchy. You can access only instrument panels and windows that were open when you exported the configuration. You can access only signals and parameters that were in loaded signal and parameter groups when you exported the configuration.

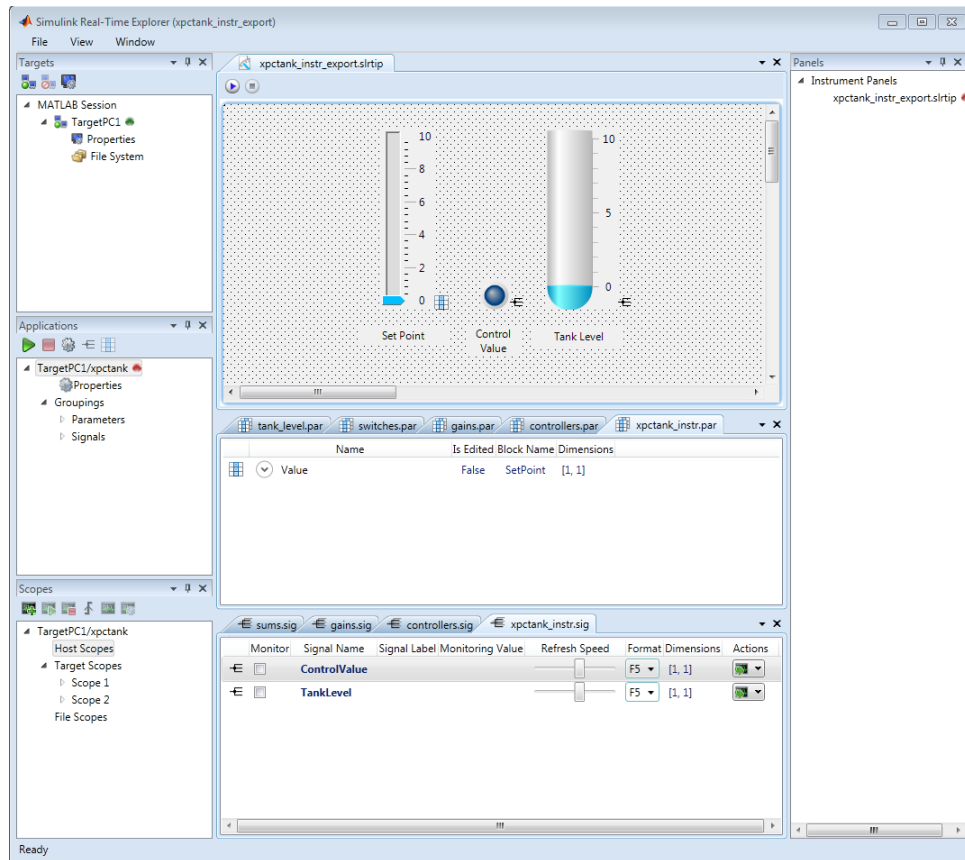
---

- 1 Copy `xpctank_instr_export.zip` from the original folder to the corresponding folder on the deployment computer, for example `C:\workdir`.
- 2 Double-click `xpctank_instr_export.zip`.
- 3 In the unzip program dialog box, click **Extract**.
- 4 Select the extraction root folder, and then click **Extract**.
- 5 Navigate into folder `xpctank_instr_export` in the extraction root folder.
- 6 Connect the target computer to the deployment computer. Restart the target computer.

Verify that the target computer loads the Simulink Real-Time kernel and the real-time application.

- 7 Double-click `xpctank_instr_export.exe`.

When the standalone executable starts, it looks like the figure.



Use the executable interface to interact with the real-time application on the target computer.

- If a signal is accessible, you can add a scope and attach the signal to the scope. Scopes that you add and remove do not change the model.
- If you remove a window, you can restore it by clicking **File > Restore Original View**.

### More About

- “Instruments and Instrument Panels”



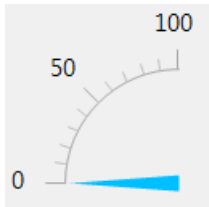
## Instruments — Alphabetical List

GaugeAngular  
GaugeFluidLevel  
GaugeLinear  
GroupBox  
HexadecimalDisplay  
Knob  
Label  
LED  
NumericDisplay  
NumericEntry  
NumericUpDownEntry  
Panel  
PictureBox  
Slider  
SwitchLED

# GaugeAngular

Graphic instrument to display signal values



## Description



Use the GaugeAngular instrument to display real-valued data suitable for an angular gauge, such as pressure, speed, and current.

## Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## Scale Graphic Display

The root node of this parameter is **Instrument**.

Parameter	Usage
AutoSize	If True, size the graphic to accommodate the parts of the display

The root node of these parameters is **Instrument+ScaleDisplay+GeneratorAuto**.

Parameter	Usage
-----------	-------

<b>DesiredIncrement</b>	Display of major tick values. number of labels = span/(desired increment + 1). Does nothing if the required labels do not fit in the space available in the graphic.
<b>FixedMinMaxMajor</b>	If True, the top and bottom ticks are constrained to be major ticks with min/max values defined by <b>Min</b> and <b>Span</b>
<b>MidIncluded</b>	If True, insert a tick halfway between major ticks.  If <b>MinorCount</b> is even, space the minor ticks equally around the center tick. If <b>MinorCount</b> is odd, replace the center tick with the middle tick. If
<b>MinorCount</b>	Number of minor ticks between major ticks
<b>MinTextSpacing</b>	Minimum space between scale ticks

## Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values FixedDecimalPoints, SignificantDigits, None
<b>Style</b>	One of the values Number, Thousands, Prefix, Exponent, Price32nds, DateTime, DateTimeUTC
<b>UnitsText</b>	Display unit next to tick labels

## General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

<b>Parameter</b>	<b>Usage</b>
<b>Min</b>	Minimum possible value
<b>Reverse</b>	If True, flip the display to increase in the opposite direction
<b>ScaleType</b>	One of the values <b>Linear</b> , <b>Log10</b> , and <b>SplitLinearLog10</b>
<b>Span</b>	Number of values between the minimum and maximum values

## **Angular Scale Range**

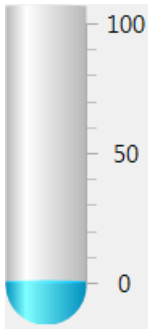
The root node of these parameters is **Instrument+ScaleRange**.

<b>Parameter</b>	<b>Usage</b>
<b>AngleMin</b>	Specify starting point of scale, from bottom of circle
<b>AngleSpan</b>	Specify number of degrees taken up by scale

# GaugeFluidLevel

Graphic instrument to display values of fluid sensor signals



## Description



Use the GaugeFluidLevel instrument to display real-valued data suitable for a fluid gauge, such as volume and pressure.

## Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## Scale Graphic Display

The root node of this parameter is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , size the graphic to accommodate the parts of the display

The root node of these parameters is **Instrument+ScaleDisplay+GeneratorAuto**.

Parameter	Usage
<b>DesiredIncrement</b>	Display of major tick values. number of labels = span / (desired increment + 1). Does nothing if the required labels do not fit in the space available in the graphic.
<b>FixedMinMaxMajor</b>	If <b>True</b> , the top and bottom ticks are constrained to be major ticks with min/max values defined by <b>Min</b> and <b>Span</b>
<b>MidIncluded</b>	If <b>True</b> , insert a tick halfway between major ticks.  If <b>MinorCount</b> is even, space the minor ticks equally around the center tick. If <b>MinorCount</b> is odd, replace the center tick with the middle tick. If
<b>MinorCount</b>	Number of minor ticks between major ticks
<b>MinTextSpacing</b>	Minimum space between scale ticks

## Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values <b>FixedDecimalPoints</b> , <b>SignificantDigits</b> , <b>None</b>
<b>Style</b>	One of the values <b>Number</b> , <b>Thousands</b> , <b>Prefix</b> , <b>Exponent</b> , <b>Price32nds</b> , <b>DateTime</b> , <b>DateTimeUTC</b>
<b>UnitsText</b>	Display unit next to tick labels

## General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

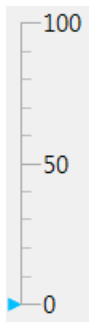
---

<b>Parameter</b>	<b>Usage</b>
<b>Min</b>	Minimum possible value
<b>Reverse</b>	If True, flip the display to increase in the opposite direction
<b>ScaleType</b>	One of the values <code>Linear</code> , <code>Log10</code> , and <code>SplitLinearLog10</code>
<b>Span</b>	Number of values between the minimum and maximum values

## GaugeLinear

Graphic instrument to display signal values



### Description



Use the GaugeLinear instrument to display real-valued data suitable for a linear gauge, such as temperature, volume, and pressure.

### Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

### Scale Graphic Display

The root node of this parameter is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , size the graphic to accommodate the parts of the display

The root node of these parameters is **Instrument+ScaleDisplay+GeneratorAuto**.



Parameter	Usage
<b>DesiredIncrement</b>	Display of major tick values. $\text{number of labels} = \text{span} / (\text{desired increment} + 1)$ . Does nothing if the required labels do not fit in the space available in the graphic.
<b>FixedMinMaxMajor</b>	If <b>True</b> , the top and bottom ticks are constrained to be major ticks with min/max values defined by <b>Min</b> and <b>Span</b>
<b>MidIncluded</b>	If <b>True</b> , insert a tick halfway between major ticks.  If <b>MinorCount</b> is even, space the minor ticks equally around the center tick. If <b>MinorCount</b> is odd, replace the center tick with the middle tick. If
<b>MinorCount</b>	Number of minor ticks between major ticks
<b>MinTextSpacing</b>	Minimum space between scale ticks

## Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values <b>FixedDecimalPoints</b> , <b>SignificantDigits</b> , <b>None</b>
<b>Style</b>	One of the values <b>Number</b> , <b>Thousands</b> , <b>Prefix</b> , <b>Exponent</b> , <b>Price32nds</b> , <b>DateTime</b> , <b>DateTimeUTC</b>
<b>UnitsText</b>	Display unit next to tick labels

## General Scale Range

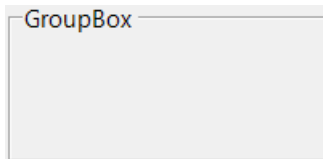
The root node of these parameters is **Instrument+ScaleRange**.

<b>Parameter</b>	<b>Usage</b>
<b>Min</b>	Minimum possible value
<b>Reverse</b>	If True, flip the display to increase in the opposite direction
<b>ScaleType</b>	One of the values <b>Linear</b> , <b>Log10</b> , and <b>SplitLinearLog10</b>
<b>Span</b>	Number of values between the minimum and maximum values

# GroupBox

Nonscrollable graphic container for instruments

## Description



The **GroupBox** graphic provides a container for other instruments. It can be stretched and shrunk at design time, but cannot be scrolled.

## Key Parameters

The key parameters are under the **Layout** node in the property list.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , the box expands at design time to make visible the instruments within it
<b>AutoSizeMode</b>	Possible values are <b>GrowAndShrink</b> and <b>GrowOnly</b> . The default is <b>GrowOnly</b> .

# HexadecimalDisplay

Text box instrument to display signal values



## Description



The **HexadecimalDisplay** instrument displays numeric data in hexadecimal format. It is used for digital data, such as status codes and register contents.

## Key Parameters

The key parameters are under the **Instrument** node in the property list.

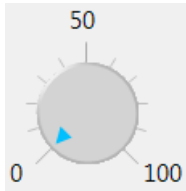
To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , the box expands at design time to make visible the specified digits. The default is <b>True</b> .
<b>DigitCount</b>	Number of hex digits to be displayed
<b>DigitLeading</b>	Possible values are <b>None</b> and <b>Zeros</b> .

# Knob

Graphic instrument to set parameter values



## Description



Use the **Knob** instrument to set real-valued data such as amplitude and frequency under conditions where an exact value is not required.

## Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## OffSwitch Graphic Display

The root node of this parameter is **Instrument+OffSwitch**.

Parameter	Usage
<b>Enabled</b>	If True, the switch is visible
<b>On</b>	If True, the switch is on

## Scale Graphic Display

The root node of this parameter is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , size the graphic to accommodate the parts of the display

The root node of these parameters is **Instrument+ScaleDisplay+GeneratorAuto**.

Parameter	Usage
<b>DesiredIncrement</b>	Display of major tick values. $\text{number of labels} = \text{span} / (\text{desired increment} + 1)$ . Does nothing if the required labels do not fit in the space available in the graphic.
<b>FixedMinMaxMajor</b>	If <b>True</b> , the top and bottom ticks are constrained to be major ticks with min/max values defined by <b>Min</b> and <b>Span</b>
<b>MidIncluded</b>	If <b>True</b> , insert a tick halfway between major ticks.  If <b>MinorCount</b> is even, space the minor ticks equally around the center tick. If <b>MinorCount</b> is odd, replace the center tick with the middle tick. If
<b>MinorCount</b>	Number of minor ticks between major ticks
<b>MinTextSpacing</b>	Minimum space between scale ticks

## Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values <b>FixedDecimalPoints</b> , <b>SignificantDigits</b> , <b>None</b>
<b>Style</b>	One of the values <b>Number</b> , <b>Thousands</b> , <b>Prefix</b> , <b>Exponent</b> , <b>Price32nds</b> , <b>DateTime</b> , <b>DateTimeUTC</b>

<b>UnitsText</b>	Display unit next to tick labels
------------------	----------------------------------

## General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

<b>Parameter</b>	<b>Usage</b>
<b>Min</b>	Minimum possible value
<b>Reverse</b>	If True, flip the display to increase in the opposite direction
<b>ScaleType</b>	One of the values <b>Linear</b> , <b>Log10</b> , and <b>SplitLinearLog10</b>
<b>Span</b>	Number of values between the minimum and maximum values

## Angular Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

<b>Parameter</b>	<b>Usage</b>
<b>AngleMin</b>	Specify starting point of scale, from bottom of circle
<b>AngleSpan</b>	Specify number of degrees taken up by scale

## Label

Graphic container for text

## Description

Label


Use the **Label** graphic to add text to the instrument layout.

## Key Parameters

The key parameters are under the **Appearance** and **Layout** nodes in the property list.

### Appearance Parameters

The root node of these parameters is **Appearance**.

Parameter	Usage
<b>Text</b>	Contains the text displayed by the label
<b>TextAlign</b>	<p>Specifies left-right, top-bottom alignment using a 3x3 matrix.</p> <p>This display represents setting TopLeft.</p> 

### Layout Parameters

The root node of this parameter is **Layout**.



---

Parameter	Usage
AutoSize	If True, size the graphic to accommodate the text

## LED

Graphic instrument to display signal values



### Description



Use the **LED** instrument to display binary (1 or 0) data.

### Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

### General Parameters

The root node of these parameters is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If True, size the graphic to accommodate the specified graphic parameters.
<b>BlinkerEnable</b>	If True, <b>LED</b> graphic blinks continuously.

### Indicator Parameters

The root node of these parameters is **Instrument+Indicator**.

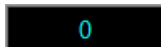
Parameter	Usage
<b>ColorActive</b>	Indicator color if signal value is 1.

Parameter	Usage
<b>ColorInactive</b>	Indicator color if signal value is 0.

# NumericDisplay

Text box instrument to display signal values



## Description



Use the **NumericDisplay** instrument to display real-valued data in selected formats.

## Key Parameters

The key parameters are under the **Instrument** and **Iocomp** nodes in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## General Parameters

The root node of this parameter is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , the box expands at design time to make visible the specified digits. The default is <b>True</b> .

## Value Display

The root node of these parameters is **Iocomp+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point

---

<b>PrecisionStyle</b>	One of the values <code>FixedDecimalPoints</code> , <code>SignificantDigits</code> , <code>None</code>
<b>Style</b>	One of the values <code>Number</code> , <code>Thousands</code> , <code>Prefix</code> , <code>Exponent</code> , <code>Price32nds</code> , <code>DateTime</code> , <code>DateTimeUTC</code>
<b>UnitsText</b>	Display unit next to tick labels

## NumericEntry

Text box instrument to set parameter values



### Description



Use the **NumericEntry** instrument to enter real-valued data in selected formats under conditions where an exact value is required.

### Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

### Text Display

The root node of these parameters is **Instrument+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values <code>FixedDecimalPoints</code> , <code>SignificantDigits</code> , <code>None</code>
<b>Style</b>	One of the values <code>Number</code> , <code>Thousands</code> , <code>Prefix</code> , <code>Exponent</code> , <code>Price32nds</code> , <code>DateTime</code> , <code>DateTimeUTC</code>
<b>UnitsText</b>	Display unit next to tick labels

# NumericUpDownEntry

Text box instrument to set parameter values



## Description



Use the **NumericUpDownEntry** instrument to enter real-valued data and increment it by a specified amount under conditions where a step change is required.

## Key Parameters

The key parameters are under the **Layout** and **Data** nodes in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## General Parameters

The root node of this parameter is **Layout**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , the box expands at design time to make visible the specified digits. The default is <b>False</b> .

## Scale Range

The root node of these parameters is **Data**.

Parameter	Usage
<b>DecimalPlaces</b>	Number of decimal places to display

<b>Increment</b>	Value to add or subtract in response to an up-arrow or down-arrow
<b>Maximum</b>	Maximum data value
<b>Minimum</b>	Minimum data value



# Panel

Scrollable graphic container for instruments

## Description



The **Panel** graphic provides a container for other instruments. You can stretch and shrink it at design time and scroll it at run time.

## Key Parameters

The key parameters are under the **Layout** node in the property list.

Parameter	Usage
<b>AutoScroll</b>	If <b>True</b> , the box scrolls at run time to make fully visible partially visible instruments within it.
<b>AutoSize</b>	If <b>True</b> , the box expands at design time to make visible the instruments within it.
<b>AutoSizeMode</b>	Possible values are <b>GrowAndShrink</b> and <b>GrowOnly</b> . The default is <b>GrowOnly</b>

## PictureBox

Graphic container for pictures



### Description



The **PictureBox** graphic provides a container for graphics, for example a photograph or line drawing.

### Key Parameters

The key parameter is under the **Behavior** node in the property list.

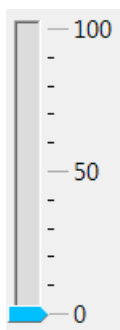
To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

Parameter	Usage
SizeMode	Possible values are Normal, StretchImage, AutoSize, CenterImage, and Zoom. The default is Normal

# Slider

Graphic instrument to set parameter values



## Description



Use the **Slider** instrument to set real-valued data such as temperature and pressure under conditions where the exact value is not required.

## Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

## Scale Graphic Display

The root node of this parameter is **Instrument**.

Parameter	Usage
<b>AutoSize</b>	If <b>True</b> , size the graphic to accommodate the parts of the display

The root node of these parameters is **Instrument+ScaleDisplay+GeneratorAuto**.

Parameter	Usage
<b>DesiredIncrement</b>	Display of major tick values. number of labels = $\text{span}/(\text{desired increment} + 1)$ . Does nothing if the required labels do not fit in the space available in the graphic.
<b>FixedMinMaxMajor</b>	If <b>True</b> , the top and bottom ticks are constrained to be major ticks with min/max values defined by <b>Min</b> and <b>Span</b>
<b>MidIncluded</b>	If <b>True</b> , insert a tick halfway between major ticks.  If <b>MinorCount</b> is even, space the minor ticks equally around the center tick. If <b>MinorCount</b> is odd, replace the center tick with the middle tick. If
<b>MinorCount</b>	Number of minor ticks between major ticks
<b>MinTextSpacing</b>	Minimum space between scale ticks

## Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
<b>Precision</b>	Number of digits to the right of the decimal point
<b>PrecisionStyle</b>	One of the values FixedDecimalPoints, SignificantDigits, None
<b>Style</b>	One of the values Number, Thousands, Prefix, Exponent, Price32nds, DateTime, DateTimeUTC
<b>UnitsText</b>	Display unit next to tick labels

## General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

---

<b>Parameter</b>	<b>Usage</b>
<b>Min</b>	Minimum possible value
<b>Reverse</b>	If True, flip the display to increase in the opposite direction
<b>ScaleType</b>	One of the values <code>Linear</code> , <code>Log10</code> , and <code>SplitLinearLog10</code>
<b>Span</b>	Number of values between the minimum and maximum values

## SwitchLED

Graphic instrument to set parameter values



### Description



Use the **SwitchLED** instrument to set a binary (1 or 0) value.

### Key Parameters

The key parameters are under the **Instrument** node in the property list.

To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon  in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots  to the right of the group.

### General Parameters

The root node of these parameters is **Instrument**.

Parameter	Usage
AutoSize	If True, size the graphic to accommodate the specified graphic parameters.
Text	Receives visible text on switch.

### Indicator Parameters

The root node of these parameters is **Instrument+Indicator**.

Parameter	Usage
ColorActive	Indicator color if signal value is 1.

Parameter	Usage
<b>ColorInactive</b>	Indicator color if signal value is 0.





# Target Computer Command-Line Interface Reference

---

## Target Computer Commands

You have a limited set of commands that you can use to work the real-time 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. Functions 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 “Control Application at Target Computer Command Line” for a description of how to use these functions and commands.

### In this section...

“Target Object Function Commands” on page 6-2

“Target Object Property Commands” on page 6-3

“Scope and Video Object Function Commands” on page 6-4

“Scope Object Property Commands” on page 6-6

“Aliasing with Variable Commands” on page 6-10

## Target Object Function Commands

When you are using the target computer command-line interface, target object functions are limited to starting and stopping the real-time 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. The target object name `tg` is used as an example for the MATLAB functions. These functions assume that you have already loaded the real-time application onto the target computer.

Target Computer Command	Description	MATLAB Equivalent
<code>start</code>	Start the real-time application currently loaded on the target computer.	<code>start(tg)</code>

Target Computer Command	Description	MATLAB Equivalent
stop	Stop the real-time application currently running on the target computer.	stop(tg)
reboot	Restart the target computer.	reboot(tg)

## 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 functions.

Target Computer Command	Description	MATLAB Equivalent
getpar param_index	Display the value of a block parameter using the parameter index.	getparam(tg, param_index)
setpar param_index = number	Change the value of a block parameter using the parameter index.	setparam(tg, param_index, number)
stoptime = number	With the value <code>number</code> , run for the specified number of seconds.	tg.StopTime = number
stoptime = Inf	With the value <code>Inf</code> , run the real-time application until you manually stop it or reset the target computer.	tg.StopTime = Inf
sampletime = number	Enter a new sample time.	tg.SampleTime = number
P#	Display the value of the block parameter with index #.	getparam(tg, param_index)

Target Computer Command	Description	MATLAB Equivalent
	For example, P2 displays the value of block parameter 2.	
S#	Display the value of the signal with index #.  For example, S2 displays the value of signal 2.	getsignal(tg, sig_index)

## Scope and Video Object Function Commands

When using the target computer command-line interface, you use scope object functions to start a scope and add signal traces. You can also collapse scopes and video displays into icons and expand them again. Notice that the functions `addscope` and `remscope` are target object functions on the development computer, 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 functions.

Target Computer Command	Description	MATLAB Equivalent
<code>addscope</code>	Without an argument, add a target scope and assign it the next available index.	<code>addscope(tg, 'target')</code>
<code>addscope scope_index</code>	With argument <code>scope_index</code> , add a target scope and assign it index <code>scope_index</code> .	<code>addscope(tg, 'target', scope_index)</code>
<code>remscope scope_index</code>	With value <code>scope_index</code> , remove scope <code>scope_index</code> .	<code>remscope(tg, scope_index)</code>
<code>remscope all</code>	With value <code>all</code> , remove all scopes.	<code>remscope(tg)</code>

Target Computer Command	Description	MATLAB Equivalent
startscope scope_index  startscope all	With value <code>scope_index</code> , start the scope with index <code>scope_index</code> .  With value <code>all</code> , start all scopes.	<code>start(sc)</code>  <code>start(getscope(tg))</code>
stopscope scope_index  stopscope all	With value <code>scope_index</code> , stop the scope with index <code>scope_index</code> .  With value <code>all</code> , stop all scopes.	<code>stop(sc)</code>  <code>stop(getscope(tg))</code>
addsignal scope_index = sig_index1, sig_index2, ...	With values <code>sig_index1</code> , <code>sig_index2</code> , ..., add the signals with these signal indexes to the scope with index <code>scope_index</code> .	<code>addsignal(sc, sig_index_vector)</code>
remsignal scope_index = sig_index1, sig_index2, ...  remsignal scope_index	With values <code>sig_index1</code> , <code>sig_index2</code> , ..., remove the signals with these signal indexes from the scope with index <code>scope_index</code> .  Without a <code>sig_index</code> value, remove all the signals from the scope with index <code>scope_index</code> .	<code>remsignal(sc, sig_index_vector)</code>  <code>remsignal(sc)</code>
show Scope scope_index	With value <code>scope_index</code> , expand scope <code>scope_index</code> from an icon.	
hide Scope scope_index	With value <code>scope_index</code> , collapse scope <code>scope_index</code> into an icon.	
show Video video_index	With value <code>video_index</code> , expand video display <code>video_index</code> from an icon.	

Target Computer Command	Description	MATLAB Equivalent
hide Video video_index	With value video_index, collapse video display video_index into an icon.	

## 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 development computer. 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 properties that you can set on the target computer. The equivalent MATLAB syntax is shown in the right column. The scope object name SC is used as an example for the MATLAB functions

Target Computer Command	Description	MATLAB Equivalent
numsamples scope_index = number	Set the number of contiguous samples captured by scope scope_index to number.	sc.NumSamples = number
decimation scope_index = 1	With value 1, the scope returns all sample points.	sc.Decimation = 1
decimation scope_index = number	With value n, the scope returns every nth sample point.	sc.Decimation = number
grid scope_index on grid scope_index off	With value on, the scope grid display is visible.  With value off, the scope grid display is not visible.	sc.Grid = 'on'  sc.Grid = 'off'
scopemode scope_index = 0	With value 0 or numerical, scope scope_index	sc.DisplayMode = 'numerical'

Target Computer Command	Description	MATLAB Equivalent
scopemode scope_index = numerical	displays signal values as text.	sc.DisplayMode = 'redraw'
scopemode scope_index = 1	With value 1 or redraw, scope scope_index plots signal values when numsamples samples has been acquired.	sc.DisplayMode = 'rolling'
scopemode scope_index = redraw		
scopemode scope_index = 3	With value 3 or rolling, scope scope_index plots signal values at every sample time.	
scopemode scope_index = rolling		
	<b>Note:</b> Value 2, sliding, will be removed in a future release. It behaves like value 3, rolling.	

Target Computer Command	Description	MATLAB Equivalent
triggermode scope_index = 0	With value 0 or freerun, scope scope_index triggers on every sample time.	sc.TriggerMode = 'freerun'
triggermode scope_index = freerun		sc.TriggerMode = 'software'
triggermode scope_index = 1	With value 1 or software, scope scope_index triggers from Command Window.	sc.TriggerMode = 'signal'
triggermode scope_index = software		sc.TriggerMode = 'scope'
triggermode scope_index = 2	With value 2 or signal, scope scope_index triggers when a designated signal changes state.	
triggermode scope_index = signal	With value 3 or scope, scope scope_index triggers when a designated scope triggers.	
triggermode scope_index = 3		
triggermode scope_index = scope		
numprepostsamples scope_index = number	Number of samples collected before or after a trigger event.	sc.NumPrePostSamples = number
triggersignal scope_index = sig_index	If triggermode is signal, triggersignal identifies the block output signal to use for triggering the scope.	sc.TriggerSignal = sig_index
triggersample scope_index = number	If triggermode is scope, triggersample specifies which sample of the triggering scope the current scope triggers on.	sc.TriggerSample = number



Target Computer Command	Description	MATLAB Equivalent
triggerlevel scope_index = number	If triggermode is signal, triggerlevel indicates the value the signal has to cross to trigger the scope to start acquiring data.	sc.TriggerLevel = number
triggerslope scope_index = 0	If triggermode is signal:  With value 0 or either, the signal triggers the scope when it crosses triggerlevel in either the rising or falling directions.  With value 1 or rising, the signal triggers the scope when it crosses triggerlevel in the rising direction.  With value 2 or falling, the signal triggers the scope when it crosses triggerlevel in the falling direction.	sc.TriggerSlope = 'Either'
triggerslope scope_index = either		sc.TriggerSlope = 'Rising'
triggerslope scope_index = 1		sc.TriggerSlope = 'Falling'
triggerslope scope_index = rising		
triggerslope scope_index = 2		
triggerslope scope_index = falling		
triggerscope scope_index = scope_index2	If triggermode is scope, triggerscope identifies the scope to use for a trigger.	sc.TriggerScope = scope_index2
triggerscopesample scope_index= integer	If triggermode is scope, triggerscopesample specifies which sample of the triggering scope to trigger on.	sc.TriggerScopeSample = integer

Target Computer Command	Description	MATLAB Equivalent
<code>ylim limit scope_index = min_y, max_y</code>	With value <code>min_y</code> , <code>max_y</code> , change the lower and upper <i>y</i> -axis values to <code>min_y</code> and <code>max_y</code> .	<code>sc.YLimit = [min_y, max_y]</code>
<code>ylim limit scope_index = auto</code>	With value <code>auto</code> , allow the lower and upper <i>y</i> -axis values to be determined by the values being displayed.	<code>sc.YLimit = 'auto'</code>

## Aliasing with Variable Commands

You can set a variable to a command string, and later use that variable to execute that command. For example, type the following on the target computer command line:

```
setvar aa = startscope 2
setvar bb = stopscope 2
```

Later, to start and stop scope 2, you can type the following:

```
aa
bb
```

The following table lists the syntax for the aliasing variable commands that you can use only on the target computer. There is no MATLAB equivalent syntax. For a usage example, see “Alias Commands at Target Computer Command Line”.

Target Computer Command	Description
<code>setvar variable_name = command</code>	Set a variable to a target computer command line string.
<code>getvar variable_name</code>	Display the value of a variable.
<code>delvar variable_name</code>	Delete a variable.
<code>delallvar</code>	Delete all variables.
<code>showvar</code>	Display a list of variables.