Simulink® Real-Time™

Reference

MATLAB® & SIMULINK®

R2015b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

Simulink® Real-Time™ Reference

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)
September 2015	Online only	Revised for Version 6.3 (Release 2015b)

Contents

1 Functions

2 Configuration Parameters

3 TLC Options Parameters

4 Instrumentation

5 Target Computer Command-Line Interface Reference

6 Simulink Real-Time Performance Advisor Checks

1

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
- "Development Computer Configuration" on page 1-12

Host-to-Target Communication

Environment Property	Description
`HostTargetComm`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of Simulink Real-Time Explorer. 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`. **Note:** 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 **Baud rate** list in the **Target Properties** pane of Simulink Real-Time Explorer.
`RS232HostPort`	Property values are `'COM1'` and `'COM2'`. Select `COM1` or `COM2` from the **Host port** list in the **Target Properties** 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
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of 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. 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.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of Simulink Real-Time Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of Simulink Real-Time Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.

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

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

<table>
<tr><th>Environment Property</th><th>Description</th></tr>
<tr><td><code>TcpIpTargetISAMemPort</code></td><td>Property value is <code>'0xnnnn'</code>.

Enter an I/O port base address in the Address box in the Target Properties pane of Simulink Real-Time Explorer.

If you are using an ISA bus Ethernet card, you must enter values for the properties <code>TcpIpISAMemPort</code> and <code>TcpIpISAIRQ</code>. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.

On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.

Set the I/O port base address to around 0x300. If one of these 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.</td></tr>
<tr><td><code>TcpIpTargetPort</code></td><td>Property value is <code>'xxxxx'</code>.

Enter a port address greater than <code>20000</code> in the Port box in the Target Properties pane of Simulink Real-Time Explorer.

This property is set by default to <code>22222</code>. The default value is higher than the reserved area (<code>telnet</code>, <code>ftp</code>, . . .) and is only of use on the target computer.</td></tr>
</table>

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is 'n', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is `0`. If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` (the default) and `'off'`. Set this value to `'off'` only if your multicore target computer is fully compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'on'`.
`MaxModelSize`	Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported. Select `1 MB` or `4 MB` from the **Model size** list in the **Target Properties** pane of Simulink Real-Time Explorer. Setting **Model size** is enabled for **Boot mode** `Stand Alone` only. 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 application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the application by the size of the DLM file produced by the build process.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default).

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

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'xxx'`, where *`xxx`* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of Simulink Real-Time Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, real-time application, data logging, and other functions that use the heap. 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 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. The Simulink Real-Time kernel can use only 4 GB of memory.
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of Simulink Real-Time Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. To use the full features of a target scope, install a keyboard on the target computer.

Environment Property	Description
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of Simulink Real-Time Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

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

Environment Property	Description
	To update the MAC address in Simulink Real-Time Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

Development Computer Configuration

Environment Property	Description
`Version`	Simulink Real-Time version number. Displayed only from `getxpcenv` when called without arguments.

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     : on
    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 `xpcinfo.txt` 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 development and target computers. At the MATLAB command prompt, type the command on the development computer.

```
xpctargetping
```

```
getxpcpci 'installed'
```

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

```
Measurement Computing    PCI-DIO24
     Bus 1, Slot 11, IRQ 10
     DI DO
     VendorID 0x1307, DeviceID 0x0028,
          SubVendorID 0x1307, SubDeviceID 0x0028
     A/D Chan: 0, D/A Chan: 0, DIO Chan: 24
     Released in: R14SP2 or Earlier

.
.
.
```

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

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

```
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 development and target computers. 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 development and target computers. 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                 IO321 (PMC-FPGA)   AI (IO321-5)...
Speedgoat                 IO331 (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 development and target computers. 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-DIO24'
       DeviceType: 'DI DO'
           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 development and target computers. 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 development and target computers. 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 development and target computers. 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 development and target computers. 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'
         DAChan: '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'
         DAChan: ''
         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"

Introduced before R2006a

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 development computer 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 development computer. 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, `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`

Introduced before R2006a

setxpcenv

Change Simulink Real-Time environment properties (not recommended)

Syntax

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

Arguments

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

Description

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

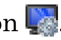
`setxpcenv` called without arguments returns a list of allowed property values in the MATLAB window.

`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`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of Simulink Real-Time Explorer. 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`. **Note:** RS-232 communication type will be removed in a future release. Use TCP/IP instead.

Environment Property	Description
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 **Baud rate** list in the **Target Properties** pane of Simulink Real-Time Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'. Select COM1 or COM2 from the **Host port** list in the **Target Properties** 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.
TcpIpGateway	Property value is 'xxx.xxx.xxx.xxx'. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of 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. 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.

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

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

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

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

Target Settings

Environment Property	Description
`EthernetIndex`	Property value is '`n`', where *n* indicates the index number for the Ethernet card on a target computer. Note that the `(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is `0`. If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` (the default) and `'off'`. Set this value to `'off'` only if your multicore target computer is fully compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'on'`.
`MaxModelSize`	Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported. Select `1 MB` or `4 MB` from the **Model size** list in the **Target Properties** pane of Simulink Real-Time Explorer. Setting **Model size** is enabled for **Boot mode** `Stand Alone` only. 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 application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the application by the size of the DLM file produced by the build process.
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default).

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

Environment Property	Description
`TargetRAMSizeMB`	Property values are `'Auto'` (the default) and `'xxx'`, where *`xxx`* is a positive value specifying the amount of RAM, in megabytes, installed on the target computer. Under **RAM size**, click the **Auto** or **Manual** button in the **Target Properties** pane of Simulink Real-Time Explorer. If you click **Manual**, enter the amount of RAM, in megabytes, installed on the target computer in the **Size(MB)** box. `TargetRAMSizeMB` defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, real-time application, data logging, and other functions that use the heap. 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 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. The Simulink Real-Time kernel can use only 4 GB of memory.
`TargetScope`	Property values are `'Disabled'` and `'Enabled'` (the default). Select or clear the **Graphics mode** check box in the **Target Properties** pane of Simulink Real-Time Explorer. If you set `TargetScope` to `Disabled`, the target computer displays information as text. To use the full features of a target scope, install a keyboard on the target computer.

Environment Property	Description
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of Simulink Real-Time Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

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

Environment Property	Description
	To update the MAC address in Simulink Real-Time Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software 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 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 Performance Advisor instead.

Benchmark execution includes generating benchmark models, building and downloading 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 μs — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.

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

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 µs — Table containing, for each target computer tested, the minimal achievable sample time for the benchmarks, in microseconds.
  - Target Information — Technical information about the target computers benchmarked.

  The entry for your target computer is highlighted.
- `xpcbench benchmark` displays the following information:
  - Benchmark name
  - Number of blocks
  - Model build time in seconds
  - Execution time in seconds
  - Minimal achievable sample time in microseconds.

`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

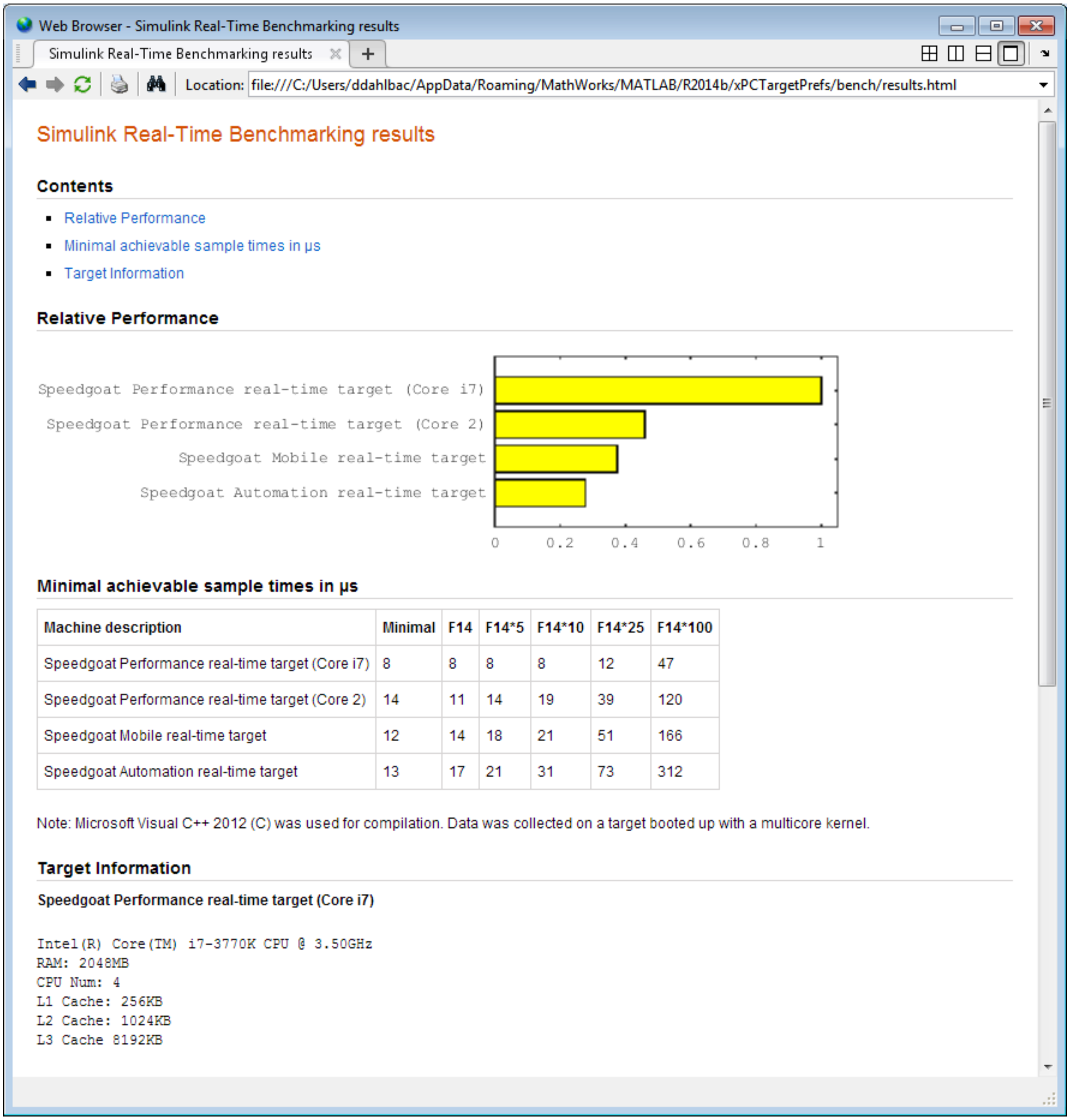
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
slrttest
```

Display representative results on predefined benchmarks.

```
xpcbench
```



Web Browser - Simulink Real-Time Benchmarking results

Simulink Real-Time Benchmarking results

Location: file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in μs
- Target Information

Relative Performance

Minimal achievable sample times in μs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
L3 Cache 8192KB
```

xpcbench this

Benchmark the target computer with the predefined benchmarks.

Start the target computer and run confidence test.

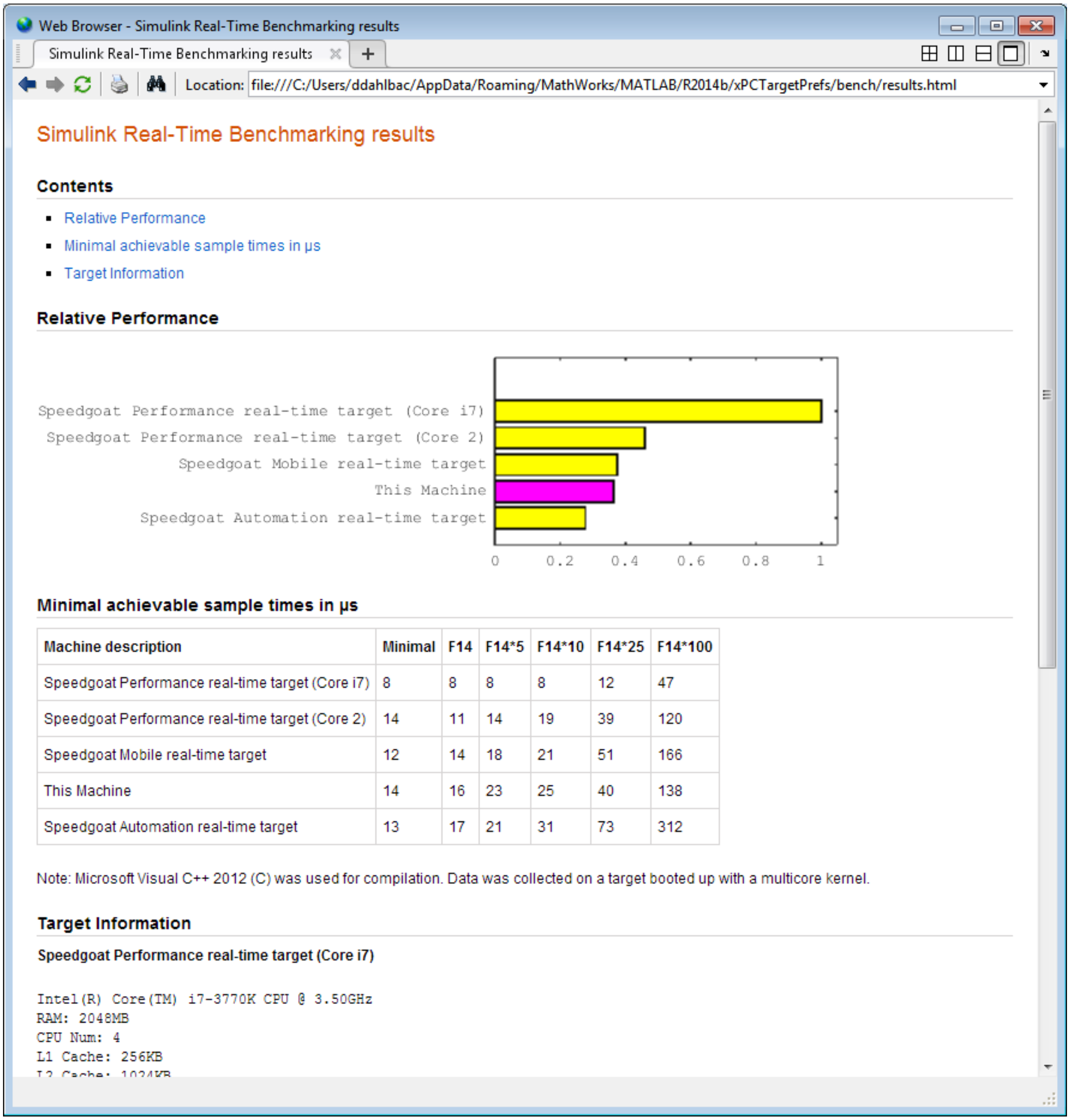
```
slrttest
```

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
```


Web Browser - Simulink Real-Time Benchmarking results

Simulink Real-Time Benchmarking results

Location: file:///C:/Users/ddahlbac/AppData/Roaming/MathWorks/MATLAB/R2014b/xPCTargetPrefs/bench/results.html

Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in µs
- Target Information

Relative Performance

Minimal achievable sample times in µs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
This Machine	14	16	23	25	40	138
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
```

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.

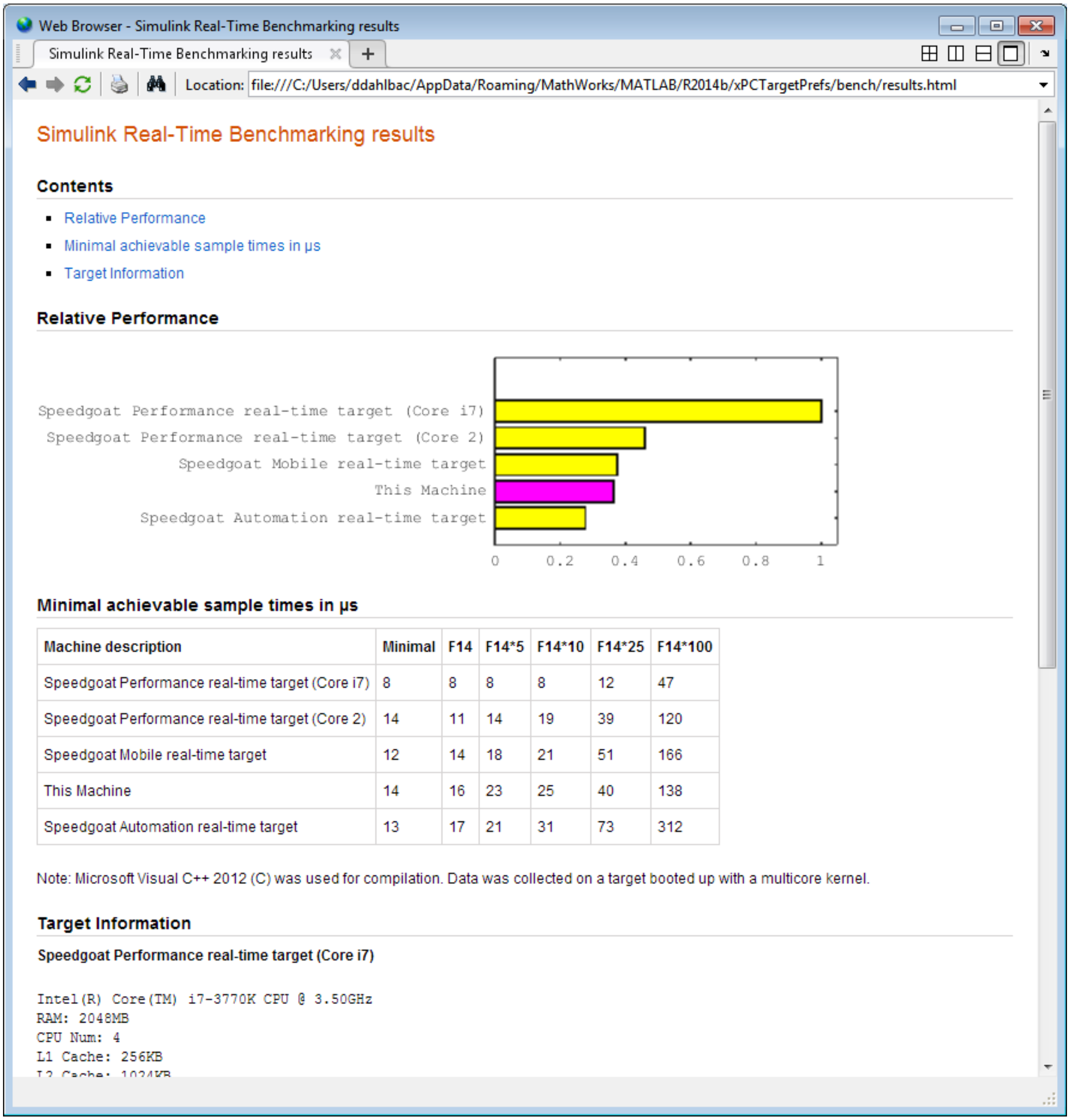
```
slrttest
```

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.
```



Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in μs
- Target Information

Relative Performance

Minimal achievable sample times in μs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
This Machine	14	16	23	25	40	138
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
```

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.

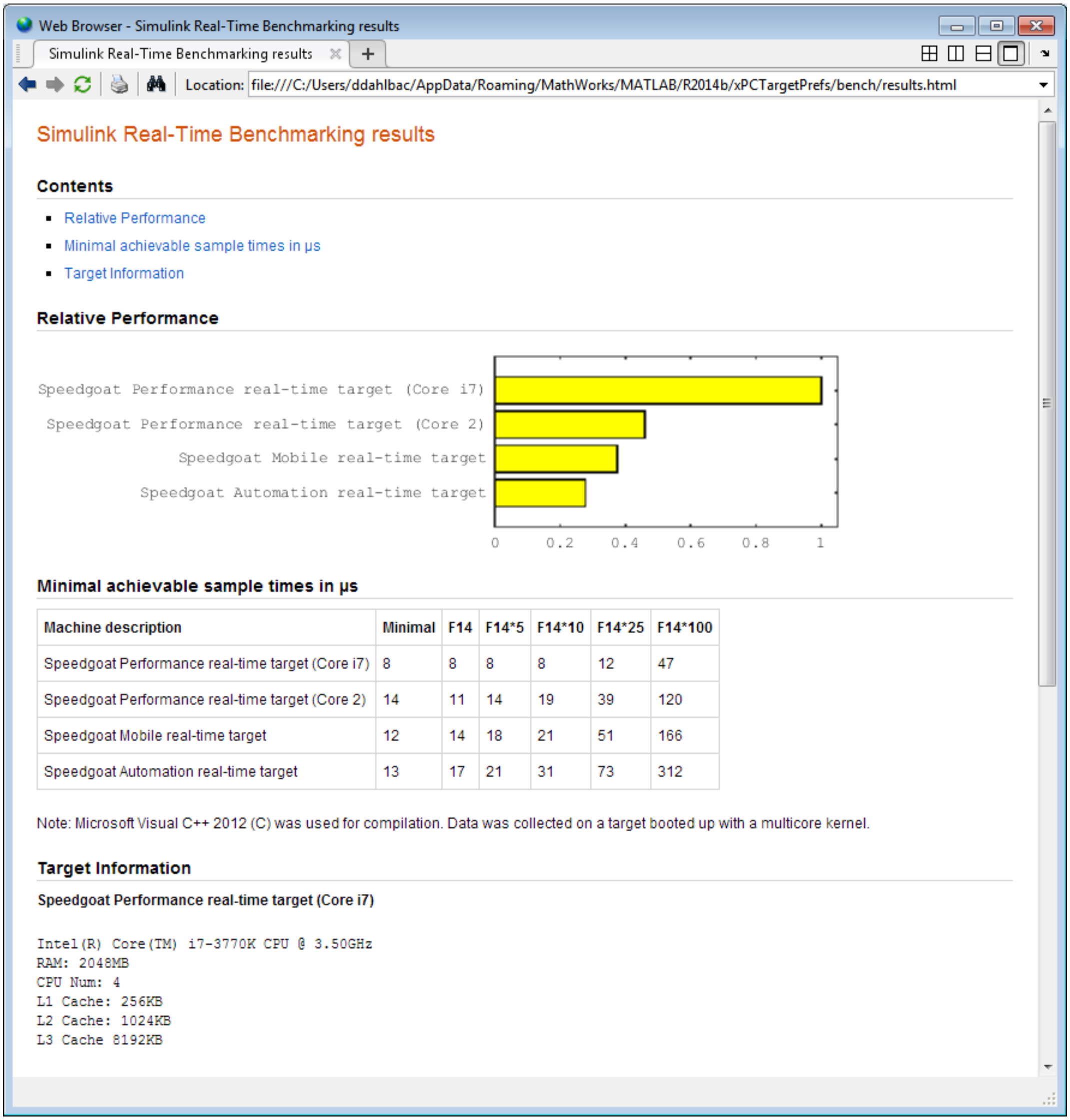
```
slrttest
```

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...'
```



Simulink Real-Time Benchmarking results

Contents

- Relative Performance
- Minimal achievable sample times in µs
- Target Information

Relative Performance

Minimal achievable sample times in µs

Machine description	Minimal	F14	F14*5	F14*10	F14*25	F14*100
Speedgoat Performance real-time target (Core i7)	8	8	8	8	12	47
Speedgoat Performance real-time target (Core 2)	14	11	14	19	39	120
Speedgoat Mobile real-time target	12	14	18	21	51	166
Speedgoat Automation real-time target	13	17	21	31	73	312

Note: Microsoft Visual C++ 2012 (C) was used for compilation. Data was collected on a target booted up with a multicore kernel.

Target Information

Speedgoat Performance real-time target (Core i7)

```
Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz
RAM: 2048MB
CPU Num: 4
L1 Cache: 256KB
L2 Cache: 1024KB
L3 Cache 8192KB
```

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`)
usermdl	Your model, *usermdl*.
`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:

Machine	Target computer information string containing CPU type, CPU speed, compiler
BenchResults	Target computer benchmark performance for all five predefined benchmarks
Desc	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:

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

More About

Tips

- Before you run `xpcbench`, you must be able to do the following:
  - Start the target computer.
  - Connect the development computer to the target computer.
  - Run the confidence test, `slrttest`, with no failures.
- After running `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` does not have continuous or discrete states. It provides an indication of the target computer interrupt latencies.
- http://www.mathworks.com/support/compilers/current_release/

See Also

`slrttest`

Introduced before R2006a

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 real-time 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

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

Description

`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 real-time applications.
- Signal acquisition
  - Create, save, and load signal groups.
  - Monitor signals.
  - Add and configure host, target, or file scopes.
  - Attach signals to or remove signals from scopes.
  - Start and stop scopes.
  - Attach signals to instruments.
- Parameter tuning
  - Create, save, and load parameter groups.
  - Display and tune parameters.
  - Attach parameters to instruments.
- Window configuration
  - Make multiple workspaces visible simultaneously.
  - Move workspaces around the window.
  - Export model configuration as a standalone executable.
  - Save and restore model configuration layouts.

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

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.

  To access more than one target computer, in the **Targets** window, configure a separate **Session** record for each target computer.

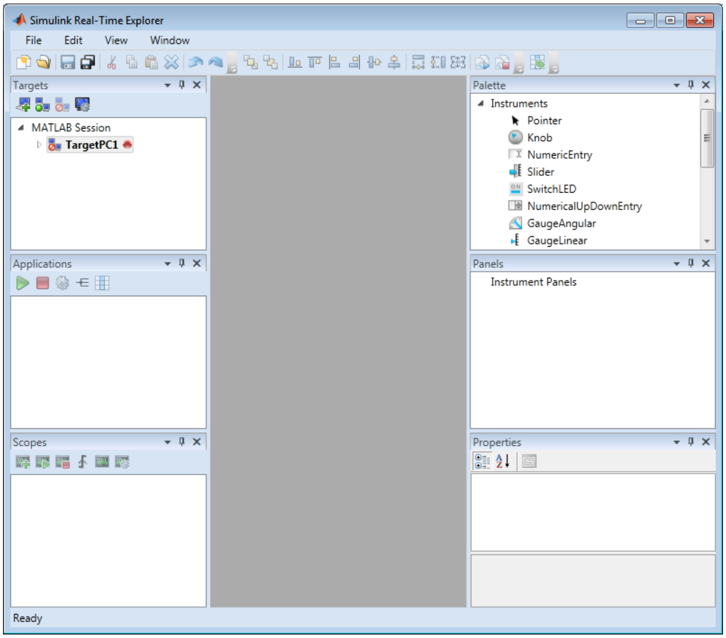
- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. To interact with multiple target computers, create separate instrument panels for each separate real-time application and target computer combination.
- If you rename a target computer, update the **TargetName** parameter for each associated instrument to maintain the connection to the real-time application.
- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.
- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the real-time application model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.
- You cannot move a signal from one signal group to another group, or create or load a new signal group.
- You can access only parameters in parameter groups that you loaded before you exported the configuration.
- You cannot move a parameter from one parameter group to another group, or create or load a new parameter group.
- You cannot save session layouts. If you close a window, you can restore the original layout using **File** > **Restore Original View**.

Examples

Default

Open Simulink Real-Time Explorer

```
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”

More About

- “Instrumentation for Real-Time Applications” on page 4-2
- “Explorer Configuration Exported to Run Outside MATLAB” on page 4-10

- "Guidelines for Exporting Explorer Configuration" on page 4-12

Introduced before R2006a

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

targetPCName	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
`xpctarget.fs`	Create file system object
`xpctarget.ftp`	Create file transfer protocol (FTP) object
`xpctarget.targets`	Create container object to manage target computer environment collection objects
`xpctarget.xpc`	Create target object representing 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`	Property values are `'RS232'` and `'TcpIp'`. Select `RS-232` or `TCP/IP` from the **Communication type** list in the **Target Properties** pane of Simulink Real-Time Explorer. 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`.

Environment Property	Description
	Note: 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, 48 00, 9600, 19200, 38400, 57600, or 115200 from the **Baud rate** list in the **Target Properties** pane of Simulink Real-Time Explorer.
RS232HostPort	Property values are 'COM1' and 'COM2'. Select COM1 or COM2 from the **Host port** list in the **Target Properties** 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
`TcpIpGateway`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the IP address for your gateway in the **Gateway** box in the **Target Properties** pane of 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. 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.
`TcpIpSubNetMask`	Property value is `'xxx.xxx.xxx.xxx'`. Enter the subnet mask of your LAN in the **Subnet mask** box in the **Target Properties** pane of Simulink Real-Time Explorer. Ask your system administrator for this value. For example, your subnet mask could be `255.255.255.0`.
`TcpIpTargetAddress`	Property value is `'xxx.xxx.xxx.xxx'`. Enter a valid IP address for your target computer in the **IP address** box in the **Target Properties** pane of Simulink Real-Time Explorer. Ask your system administrator for this value. For example, `192.168.0.10`.

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

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

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

Target Settings

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

Environment Property	Description
	`(n-1)th` Ethernet card on the target computer has an index number `'n'`. The default index number is `0`. If the target computer has multiple Ethernet cards, you must select one of the cards for the Ethernet link. This option returns the index number of the card selected on the target computer upon booting.
`LegacyMultiCoreConfig`	Property values are `'on'` (the default) and `'off'`. Set this value to `'off'` only if your multicore target computer is fully compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to `'on'`.
`MaxModelSize`	Supported property values are '1MB' (the default) and '4MB'. Value '16MB' is not supported. Select `1 MB` or `4 MB` from the **Model size** list in the **Target Properties** pane of Simulink Real-Time Explorer. Setting **Model size** is enabled for **Boot mode** `Stand Alone` only. 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 application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the application by the size of the DLM file produced by the build process.

Environment Property	Description
`MulticoreSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Multicore CPU** check box in the **Target Properties** pane of Simulink Real-Time Explorer. If your target computer has multicore processors, set this value to `'on'` to take advantage of these processors for background tasks. Otherwise, set this value to `'off'`.
`Name`	Target computer name.
`NonPentiumSupport`	Property values are `'on'` and `'off'` (the default). Select or clear the **Target is a 386/486** check box in the **Target Properties** pane of Simulink Real-Time Explorer. Set this value to `'on'` if your target computer has a 386 or 486 compatible processor. Otherwise, set it to `'off'`. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.
`SecondaryIDE`	Property values are `'on'` and `'off'` (the default). Select or clear the **Secondary IDE** check box in the **Target Properties** pane of Simulink Real-Time Explorer. Set this value to `'on'` only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to `'off'`.
`ShowHardware`	Property values are `'on'` and `'off'` (the default). If you create a target boot kernel when `ShowHardware` is `'on'` and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor. The development computer cannot communicate with the target computer after the kernel boots with `ShowHardware` set.

<table>
<tr><th>Environment Property</th><th>Description</th></tr>
<tr><td><code>TargetRAMSizeMB</code></td><td>Property values are <code>'Auto'</code> (the default) and <code>'xxx'</code>, where <code>xxx</code> is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.

Under RAM size, click the Auto or Manual button in the Target Properties pane of Simulink Real-Time Explorer. If you click Manual, enter the amount of RAM, in megabytes, installed on the target computer in the Size(MB) box.

<code>TargetRAMSizeMB</code> 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.

If <code>TargetRAMSizeMB</code> is assigned <code>'Auto'</code>, the real-time application reads the target computer BIOS and determines the amount of memory up to a maximum of 4 GB. If the 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.

The Simulink Real-Time kernel can use only 4 GB of memory.</td></tr>
<tr><td><code>TargetScope</code></td><td>Property values are <code>'Disabled'</code> and <code>'Enabled'</code> (the default).

Select or clear the Graphics mode check box in the Target Properties pane of Simulink Real-Time Explorer.

If you set <code>TargetScope</code> to <code>Disabled</code>, the target computer displays information as text.

To use the full features of a target scope, install a keyboard on the target computer.</td></tr>
</table>

Environment Property	Description
`USBSupport`	Property values are `'on'` (the default) and `'off'`. Select or clear the **USB Support** check box in the **Target Properties** pane of Simulink Real-Time Explorer. Set this value to `'on'` if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to `'off'`.

Boot Configuration

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

Environment Property	Description
	To update the MAC address in Simulink Real-Time Explorer, first click the **Reset** button in the **Target Properties** pane. You can then click the **Specify new MAC address** button to enter a MAC address manually in the **MAC address** box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.

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.

The following limitations hold:

- At most eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.

Constructor

Constructor	Description
`xpctarget.fs`	Create file system object

Methods

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

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

These methods are specific to class `fs`.

Method	Description
`xpctarget.fs.diskinfo`	Information about target computer drive
`xpctarget.fs.fclose`	Close open target computer file(s)
`xpctarget.fs.fileinfo`	Target computer file information
`xpctarget.fs.filetable`	Information about open files in target computer file system
`xpctarget.fs.fopen`	Open target computer file for reading
`xpctarget.fs.fread`	Read open target computer file
`xpctarget.fs.fwrite`	Write binary data to open target computer file
`xpctarget.fs.getfilesiz`	Size of file on target computer
`xpctarget.fs.removefile`	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

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

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

Description

`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

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

Description

Method of `xpctarget.fs` objects. From the 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

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

Description

Method of `xpctarget.fs` objects. From the 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

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

Description

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

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`).

The following limitations hold:

- At most eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.

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

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

Description

`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

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

Description

Method of xpctarget.fs objects. From the 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

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

Description

Method of `xpctarget.fs` objects. From the 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

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

Description

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

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

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

Description

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

Note: 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.

The following limitations hold:

- At most eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.

Methods

These methods are inherited by the derived classes.

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

xpctarget.fsbase.cd

Change folder on target computer (not recommended)

Syntax

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

Arguments

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

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the 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.fsbase.dir

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

Note: Method `xpctarget.fsbase.dir` 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

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

Description

Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the 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

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

Description

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

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

Methods

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

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

These methods are specific to class `ftp`.

Method	Description
`xpctarget.ftp.get`	Retrieve copy of requested file from target computer

Method	Description
`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

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

Description

Method of `xpctarget.ftp` objects. Copies the specified filename from the target computer to the current folder of the 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

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

Description

Method of xpctarget.ftp objects. Copies a file from the 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
`xpctarget.targets`	Create container object to manage target computer environment collection objects

Methods

Method	Description
`xpctarget.targets.Add (env collection object)`	Add a new Simulink Real-Time environment collection object.
`xpctarget.targets.getTargetNames (env collection object)`	Retrieve the Simulink Real-Time environment collection object names.
`xpctarget.targets.Item (env collection object)`	Retrieve Simulink Real-Time environment collection object.
`xpctarget.targets.makeDefault (env collection object)`	Set target computer environment collection object as default.
`xpctarget.targets.Remove (env collection object)`	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 `xpctarget.env` object that references the default target computer object environment.	No
`NumTargets`	Returns the number of target computer environment objects in the container.	No
`ShowTargets`	When `on` (the default) shows information about the targets. When `off`, 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
`xpctarget.xpc`	Create target object representing real-time application

Methods

Method	Description
`xpctarget.xpc.addscope`	Create scopes
`xpctarget.xpc.close`	Close serial port connecting development computer with target computer
`xpctarget.xpc.getlog`	All or part of output logs from target object
`xpctarget.xpc.getparam`	Value of target object parameter index
`xpctarget.xpc.getparami`	Parameter index from parameter list
`xpctarget.xpc.getparamn`	Block path and parameter name from index list
`xpctarget.xpc.getscope`	Scope object pointing to scope defined in kernel
`xpctarget.xpc.getsignal`	Value of target object signal index
`xpctarget.xpc.getsignal`	Signal index or signal property from signal list
`xpctarget.xpc.getsignal`	Return vector of signal indices

Method	Description
`xpctarget.xpc.getsignal`	Return signal label
`xpctarget.xpc.getsignal`	Signal name from index list
`xpctarget.xpc.load`	Download real-time application to target computer
`xpctarget.xpc.loadparam`	Restore parameter values saved in specified file
`xpctarget.xpc.reboot`	Reboot target computer
`xpctarget.xpc.remscope`	Remove scope from target computer
`xpctarget.xpc.saveparam`	Save current real-time application parameter values
`xpctarget.xpc.setparam`	Change writable target object parameters
`xpctarget.xpc.start (real-time application object)`	Start execution of real-time application on target computer
`xpctarget.xpc.stop (real-time application object)`	Stop execution of real-time application on target computer
`xpctarget.xpc.targetpin`	Test communication between development and target computers
`xpctarget.xpc.unload`	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 real-time application, which is identical to the name of the Simulink model that you built the application from.	No
`AvgTET`	Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The TET includes: • Complete I/O latency. • Data logging (the parts that happen in a real-time task). This includes data captured in scopes. • Asynchronous interruptions. • Parameter updating latency (if the **Double buffer parameter changes** parameter is set in the **Simulink Real-Time Options** node of the model Configuration Parameters dialog box). Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are: • Time required to measure TET	No

Property	Description	Writable
	• Interrupt latency required to schedule and run one step of the model	
`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: • Time-equidistant logging. Logs a data point at every time interval. Set value to `'Normal'`. • Value-equidistant logging. Logs a data point only when an output signal from the `OutputLog` changes by a specified value (increment). Set the value to the difference in signal values.	Yes

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

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'`: • Property value. Value of the parameter in a Simulink block. • Type. Data type of the parameter. Always `double`. • Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix. • Parameter name. Name of a parameter in a Simulink block. • Block name. Name of a Simulink block.	No
`SampleTime`	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See "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'`. • Property name. `S0`, `S1`. . . • Property value. Value of the signal. • Block name. Name of the Simulink block the signal is from.	No
`StateLog`	Storage in the MATLAB workspace for the state or x-vector logged during execution of the 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 **Solver** pane of the Configuration Parameters dialog box. When the `ExecTime` reaches `StopTime`, the application stops running.	Yes
`TETLog`	Storage in the MATLAB workspace for a vector containing task execution times during execution of the real-time application. To enable logging of the TET, you need to select the **Log Task Execution Time** check box in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box.	No
`TimeLog`	Storage in the MATLAB workspace for the time or T-vector logged during execution of the real-time application.	No

Property	Description	Writable
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

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.

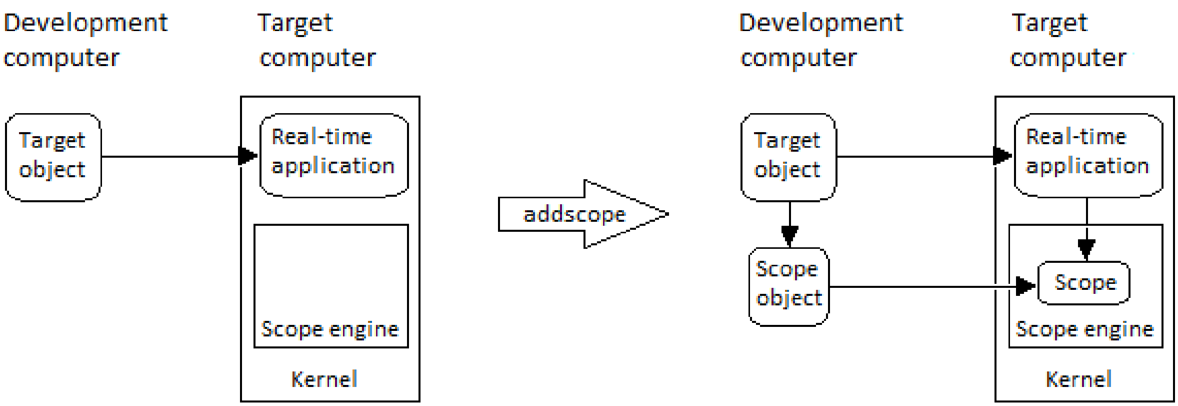
```
addscope
addscope scope_number
```

Arguments

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

Description

`addscope` creates a scope of the specified type and updates the target object property `Scopes`. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The 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 `n`th sample point. This argument is optional. Default is `1`.

Description

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

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)
```

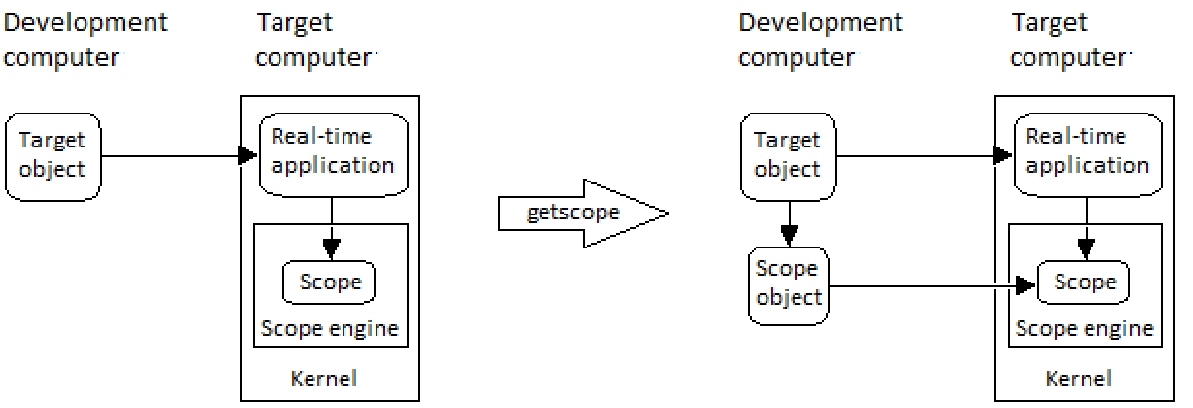
Arguments

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

Description

`getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can 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

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

Description

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

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

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

Description

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

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,real_time_application)`

Description

`target_object = load(target_object,real_time_application)` loads application `real_time_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. `real_time_application` can also contain the absolute path to the real-time application, without file extension.

You must build the real-time 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 standalone real-time application, you must rebuild the standalone application files with the new real-time application and transfer the updated files to the target computer using `xpctarget.ftp`. Then, restart the target computer with the new real-time 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 real-time 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

`target_object`	Name of an existing target object.
`filename`	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`.

For some target computers, you can also use this method to restart the target computer back to Windows® after removing the target boot disk.

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'
```

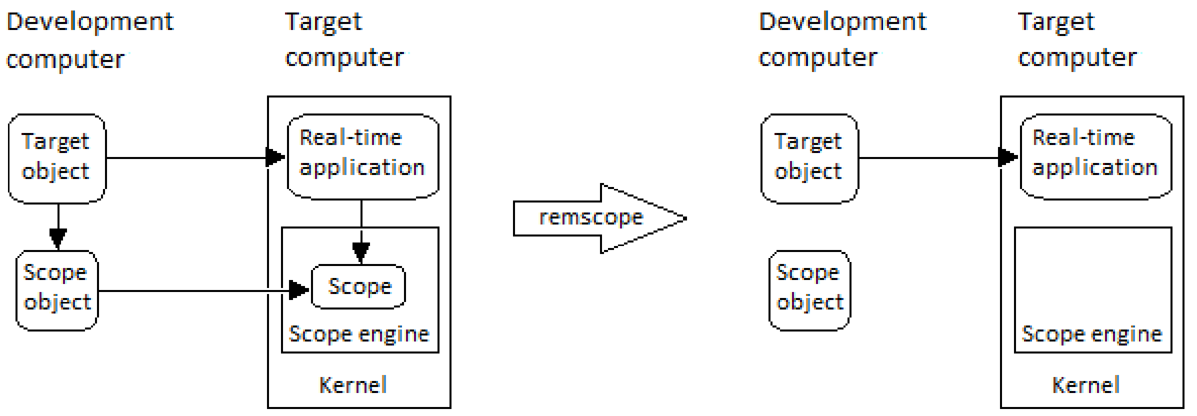
Arguments

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

Description

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

`target_object`	Name of an existing target object.
`filename`	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 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)` | `xpctarget.xpc.load` | `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 `slrtpingtarget`or 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 standalone real-time application, you must rebuild the standalone application with the new real-time application, and 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

`xpctarget.xpc.load` | `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.

The following limitations hold:

- At most eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.

Methods

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

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

Properties

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

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

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

Property	Description	Writable
`TriggerScope`	If `TriggerMode` is `'Scope'`, identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property `TriggerScope` to the scope index of the master scope.	Yes
`TriggerSignal`	If `TriggerMode` is `'Signal'`, identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property `Signal`.	Yes
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the development computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

These properties are specific to class `xpcfs`.

Property	Description	Writeable
`AutoRestart`	Values are `'on'` and `'off'`. For file scopes, enable the file scope to collect data up to the number of samples (`NumSamples`), then start over again, appending the new data to the end of the signal data file. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples**, then stop. If the named signal data file already exists when you start the real-time	No

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

Property	Description	Writeable

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

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

Property	Description	Writeable
`WriteMode`	For file scopes, specify when a file allocation table (FAT) entry is updated. Values are `'Lazy'` or `'Commit'`. Both modes write the signal data to the file. With `'Commit'` mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system 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). For host or target scopes, this parameter has no effect.	Yes
`WriteSize`	Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (`NumSamples`), collect data in multiples of `WriteSize`. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. If you experience a system crash, you can expect to lose an amount of data the size of `WriteSize`. For host or target scopes, this parameter has no effect.	Yes

xpctarget.xpcsc.addsignal

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

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

Description

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

Note:

- You must stop the scope before you can add a signal to it.
- 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.xpcsctg Class`. All methods and properties are inherited by the derived classes. When a mixture of derived classes are stored in a scope collection, only the base class methods and properties are available. 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 then 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
`xpctarget.xpcsc.addsign`	Add signals to scope represented by scope object
`xpctarget.xpcsc.remsign`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software trigger start of data acquisition for scope(s)

Properties

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 `TriggerMode` to `'FreeRun'`, this property has no effect on data acquisition.	Yes

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

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

Property	Description	Writable
`Type`	Determines whether the scope is displayed on the development computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	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

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

Description

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

Note:

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

- 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

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

Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the 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 `getscope`, which returns a `scope_object` vector.
`signal_index_vector`	Index for a single scope or list of scope indices in vector form.
`scope_index`	Single scope index.

Description

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

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

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

Description

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

Note: 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.xpcschost 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 development computer. The development computer displays the data using a scope viewer or other MATLAB functions.

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

Methods

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

Method	Description
`xpctarget.xpcsc.addsign`	Add signals to scope represented by scope object
`xpctarget.xpcsc.remsign`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software trigger start of data acquisition for scope(s)

Properties

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

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No

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

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

Property	Description	Writable
`TriggerSlope`	If `TriggerMode` is `'Signal'`, indicates whether the trigger is on a rising or falling signal. Values are `'Either'` (default), `'Rising'`, and `'Falling'`.	Yes
`Type`	Determines whether the scope is displayed on the development computer or on the target computer. Values are `'Host'`, `'Target'`, and `'File'`. Property `Type` is set only once, when the scope is created on the target computer.	No

These properties are specific to class `xpcschost`.

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

xpctarget.xpcsctg 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.xpcsctg` will be removed in a future release. Use class `SimulinkRealTime.targetScope` instead.

Methods

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

Method	Description
`xpctarget.xpcsc.addsign`	Add signals to scope represented by scope object
`xpctarget.xpcsc.remsign`	Remove signals from scope represented by scope object
`xpctarget.xpcsc.start (scope object)`	Start execution of scope on target computer
`xpctarget.xpcsc.stop (scope object)`	Stop execution of scope on target computer
`xpctarget.xpcsc.trigger`	Software trigger start of data acquisition for scope(s)

Properties

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

Property	Description	Writable
`Application`	Name of the Simulink model associated with this scope object.	No

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

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

Property	Description	Writable
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the development computer or on the target computer. Values are 'Host', 'Target', and 'File'. Property Type is set only once, when the scope is created on the target computer.	No

These properties are specific to class xpcsctg.

Property	Description	Writeable
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'. For host or file scopes, this parameter has no effect. .	Yes
Grid	Values are 'on' and 'off'. For host or file scopes, this parameter has no effect.	Yes
Mode	**Note:** The Mode property will be removed in a future release. • For target scopes, use DisplayMode. • For file scopes, use WriteMode. • For host scopes, this parameter has no effect.	Yes

Property	Description	Writeable
`YLimit`	Minimum and maximum *y*-axis values. This property can be set to `'auto'`. For host or file scopes, this parameter has no effect.	Yes

xpctargetping

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`

Introduced before R2006a

xpctargetspy

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

Syntax

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

Arguments

`target_object`	Variable name to reference the target object.
`target_object_name`	Target object name as specified in the 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 development computer display with another target display, 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 development computer 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 computer 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 a real-time application run.

`xpctest('noreboot')` or `xpctest('-noreboot')` skips the reboot test on the default target computer. Use this option if target computer 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 computer 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 MATLAB interface (not recommended)

Syntax

```
xpcwwwenable
xpcwwwenable('target_obj_name')
```

Description

`xpcwwwenable` disconnects the real-time application from the current 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.

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

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"

Build for default target computer

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 computer name** field so that you can enter the target computer to which to download the real-time application.

Dependency

When cleared, this parameter enables **Specify target computer name**.

Command-Line Information

Parameter: `xPCisDefaultEnv`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

See Also

- “Ethernet Link Setup”
- “Serial Link Setup”

Specify target computer 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, clear **Download to default target computer**.

Command-Line Information

Parameter: `xPCTargetPCEnvName`
Type: string
Value: Any valid target computer
Default: `''`

See Also

"Simulink Real-Time Explorer Basic Operations"

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"

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: `RL32ObjectName`
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

☑ On

Waits the default amount of seconds (5) for the real-time application to be downloaded to the target computer.

☐ Off

Enables the **Specify the communication timeout in seconds** field so that you can enter the maximum length of time in seconds you want to wait for a 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 execution mode of downloaded code.

Settings

Default: `Real-Time`

`Real-Time`

Executes downloaded code as a real-time application.

`Freerun`

Executes downloaded code 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® DSCP-200/300 board.

`Quatech ESC-100`

Specifies that the interrupt source is the Quatech ESC-100 board.

`Quatech QSC-100`

Specifies that the interrupt source is the Quatech QSC-100 board.

`Quatech QSC-200/300`

Specifies that the interrupt source is the Quatech QSC-200/300 board.

`RTD_DM6804`

Specifies that the interrupt source is the Real-Time Devices DM6804 board.

`SBS_25x0_ID_0x100`

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x100.

`SBS_25x0_ID_0x101`

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x101.

`SBS_25x0_ID_0x102`

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x102.

`SBS_25x0_ID_0x103`

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x103.

`Scramnet_SC150+`

Specifies that the interrupt source is the Systran® Scramnet+ SC150 board.

`Softing_CAN-AC2-104`

Specifies that the interrupt source is the Softing® CAN-AC2-104 board.

`Softing_CAN-AC2-PCI`

Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

`Speedgoat_IO301`

Specifies that the interrupt source is the Speedgoat IO301 FPGA board.

`Speedgoat_IO302`

Specifies that the interrupt source is the Speedgoat IO302 FPGA board.

`Speedgoat_IO303`

Specifies that the interrupt source is the Speedgoat IO303 FPGA board.

`Speedgoat_IO311`

Specifies that the interrupt source is the Speedgoat IO311 FPGA board.

`Speedgoat_IO312`

Specifies that the interrupt source is the Speedgoat IO312 FPGA board.

`Speedgoat_IO313`

Specifies that the interrupt source is the Speedgoat IO313 FPGA board.

`Speedgoat_IO314`

Specifies that the interrupt source is the Speedgoat IO314 FPGA board.

`Speedgoat_IO321`

Specifies that the interrupt source is the Speedgoat IO321 FPGA board.

`Speedgoat_IO331`

Specifies that the interrupt source is the Speedgoat IO331 FPGA board.

`UEI_MFx`

Specifies that the interrupt source is a United Electronic Industries UEI-MF series board.

`None/Other`

Specifies that the I/O board has no interrupt source.

Command-Line Information

Parameter: `xPCIRQSourceBoard`
Type: string
Value: `'ATI-RP-R5' |`
`'AudioPMC+' |`
`'Bitflow NEON' |`
`'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_IO301' |
'Speedgoat_IO302' |
'Speedgoat_IO303' |
'Speedgoat_IO311' |
'Speedgoat_IO312' |
'Speedgoat_IO313' |
'Speedgoat_IO314' |
'Speedgoat_IO321' |
'Speedgoat_IO331' |
'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 profiling events (each uses 20 bytes)

Enter the maximum of events to log for the profiling tool.

Settings

Default: 5000

The maximum number of events to be logged for the profiling tool.

Tips

- An event is the start of end of an interrupt or iteration of the model. For example, one sample can four events: the beginning and end of an interrupt, and the beginning and end of an iteration.
- 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: `xpcDblBuff`
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® 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”

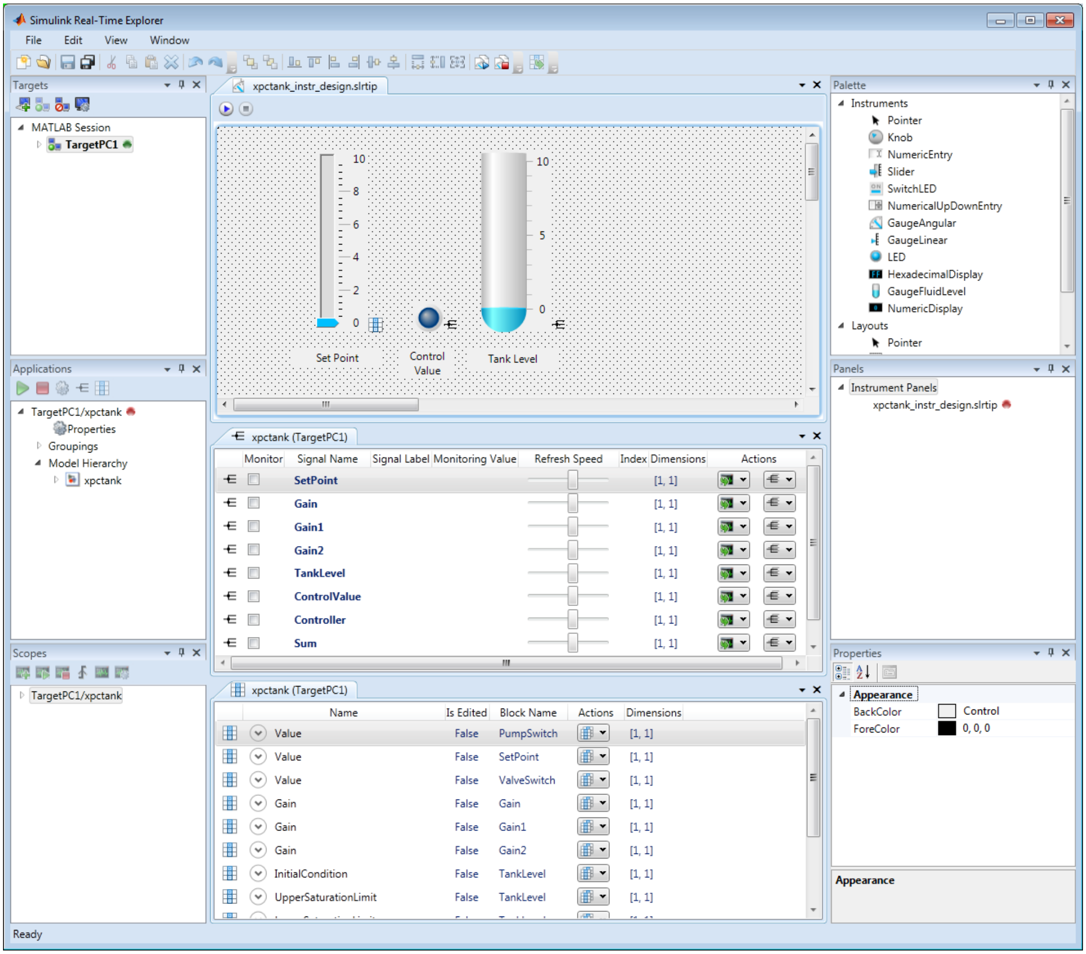
Instrumentation

Instrumentation for Real-Time Applications

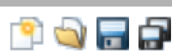
In this section...

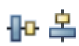
To visualize the behavior of a real-time application running on a target computer, Simulink Real-Time Explorer provides instrument panels that you can use. An instrument panel is an Explorer workspace into which you can insert one or more instruments. You can create and load panels from the toolbar, from the File menu, and from the Panels window. You can display simultaneously as many panels as can fit on the screen.

After creating one or more instrument panels, you can drag instruments to the panels and drag parameters and signals to the instruments. You can add layout elements to clarify the relationship between the instruments and the model. You can then start your real-time application from Simulink Real-Time Explorer and start the instrument panels to control the parameters and view the signal outputs.



You can manipulate the instruments using the toolbar icons.

Action	Icon	Notes
New, Open, Save, SaveAll		The available operations vary with the active window. • You can save an instrument panel from

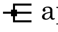
Action	Icon	Notes
		within only the window for that panel. • You can create or open a signal or parameter group from within only the **Applications** window. • You can save a group from within only the window for that group. • You can create or open an instrument panel regardless of active window. • You can use the SaveAll button regardless of active window.
Cut, Copy, Paste, Delete		Applies to instruments only.
Layer		Applies to instruments only.
Align on edges		Applies to instruments only.
Align on centers		Applies to instruments only.
Equalize sizes		Applies to instruments only.
Undo and redo		Available after unsaved change only.
Run, Stop, Run all, Stop all		Run and stop all active instrument panels.

Instrument Selection and Binding

To instrument a model, populate an instrument panel with instruments compatible with the parameters and signals that they represent in the model.

To make an instrument interact with the real-time application, bind a signal or parameter to the instrument. You bind the signal or parameter by dragging it from the signal or parameter viewer to the instrument. You can also drag a signal or parameter from a signal or parameter group to an instrument.

You can bind a signal to a signal display instrument but not to a parameter tuning instrument. You can bind a parameter to a parameter tuning instrument and to a signal display instrument.

When you bind a signal to a signal display instrument, the Signal icon  appears next to the instrument.

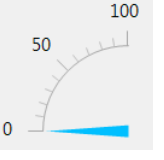
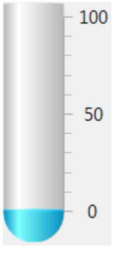
When you bind a parameter to a parameter tuning or signal display instrument, the Parameter icon  appears next to the instrument.

When the instrument panel is running, you can tune parameters using parameter tuning instruments. The software transmits the parameter changes to the real-time application. You can view the changed behavior using signal display instruments.

When the instrument panel is not running, you can add, remove, and lay out instruments and connect signals and parameters to them.

Signal and Parameter Display Instruments

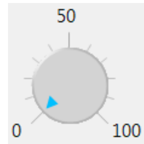
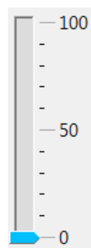
The table shows some of the tasks you can do with signal and parameter display instruments.

To	Requiring	Use
• Show the pressure in a container. • Show the speed of a vehicle. • Show current or voltage in a circuit.	• Real-valued data • Analogy to physical instruments that show approximate value by angular displacement	 `GaugeAngular`
• Show the level of fluid in a container. • Show the pressure in a pipe.	• Real-valued data • Analogy to physical instruments that show approximate value by vertical displacement	

To	Requiring	Use
• Show the pressure in a container. • Show audio output power. • Show current or voltage in a circuit.	• Real-valued data • Analogy to physical instruments that show approximate value by linear displacement	GaugeFluidLevel  GaugeLinear
• Show traffic on a digital bus. • Show the state of a state machine.	• Integer-valued data • Analogy to physical instruments that show hexadecimal values	 HexadecimalDisplay
• Show on-off state of a switch. • Show on-off state of a bidirectional pin.	• Boolean data • Analogy to physical instruments that show value by lights turning on and off	 LED
• Show a temperature measurement to given precision. • Show a voltage measurement to given precision. • Show a date and time.	• Real-valued data • Analogy to physical instruments that show real values in decimal and other format.	 NumericDisplay

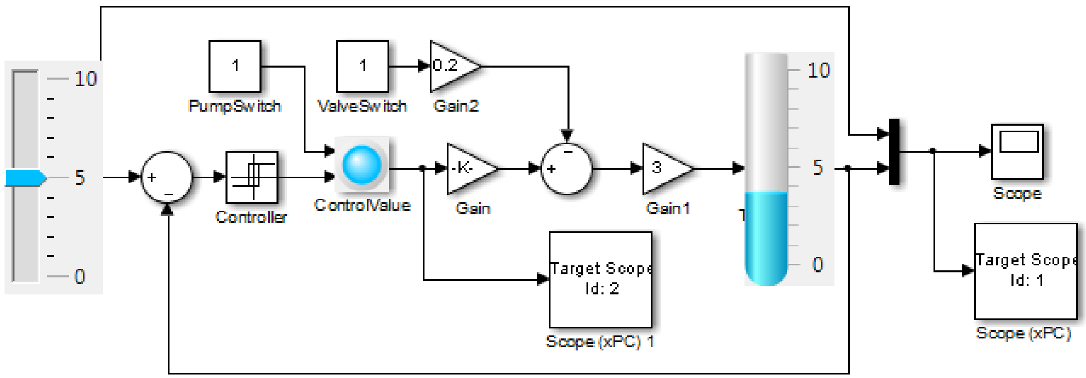
Parameter Tuning Instruments

The table shows some of the tasks you can do with parameter tuning instruments.

To	Requiring	Use
• Control gain of a radio receiver. • Control amplification of a radio transmitter.	• Real-valued data • Analogy to physical controls that set approximate value by angular displacement	 `Knob`
• Enter initial set point control value for thermostat. • Enter seed value for random number generator.	• Real-valued data • Analogy to physical controls that set exact numeric value	 `NumericEntry`
• Control car radio audio volume using step-increment. • Smoke test controller range in small number of clicks.	• Real-valued data • Analogy to physical controls that set initial value and step increment	 `NumericUpDownEntry`
• Control frequency of a radio receiver. • Control pressure valve setting.	• Real-valued data • Analogy to physical controls that set approximate values by linear displacement	 `Slider`
• Turn on a power supply. • Close a gate valve.	• Boolean data • Analogy to physical controls that are either on or off	 `SwitchLED`

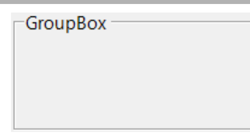
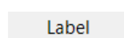
Layout Elements

To make the relationship between the instruments and the model clearer, you can add layout elements, such as a picture box. The figure shows a picture box that shows which signals the instruments are bound to.



Tank Level Control System

The table shows some ways that you can modify your layout with layout elements.

To	Requiring	Use
• Group electrical instruments together and label the group. • Group pressure instruments together and label the group.	• Design-time box resizing • Run-time static display	 `GroupBox`
• Label an electrical instrument. • Label a pressure instrument.	• Design-time box resizing, left-right and up-down text alignment • Run-time static display	 `Label`
• Group the electrical group with the pressure group and scroll between the groups.	• Design-time box resizing • Run-time box scrolling	

To	Requiring	Use
• Group a picture box with a group of instruments to show what the instruments are measuring.		`Panel`
• Insert an image of a circuit diagram on a panel behind electrical instruments. • Insert an image of a circulation diagram on a panel behind pressure instruments.	• Design-time image stretch, zoom, center, and autosize • Run-time static display	 `PictureBox`

Explorer Configuration Exported to Run Outside MATLAB

You can export a Simulink Real-Time Explorer configuration as a standalone executable to run outside MATLAB on a computer compatible with Windows. The standalone Explorer supports a subset of the capabilities that it supports under MATLAB.

- You cannot change the communication parameters that the interface uses to communicate with the target computer. Before you export the Simulink Real-Time Explorer configuration, configure and test the communication parameters.

  To access more than one target computer, in the **Targets** window, configure a separate **Session** record for each target computer.
- For each instrument, the exporting software records the real-time application and target computer environment with which it is associated. To interact with multiple target computers, create separate instrument panels for each separate real-time application and target computer combination.
- If you rename a target computer, update the **TargetName** parameter for each associated instrument to maintain the connection to the real-time application.
- You cannot load or unload a real-time application from the standalone executable. Before you start the executable, start the real-time application on the target computer.
- You can access only instrument panels and windows that you loaded before you exported the configuration.
- You cannot access the real-time application model hierarchy from the standalone executable.
- You can access only signals in signal groups that you loaded before you exported the configuration.
- You cannot move a signal from one signal group to another group, or create or load a new signal group.
- You can access only parameters in parameter groups that you loaded before you exported the configuration.
- You cannot move a parameter from one parameter group to another group, or create or load a new parameter group.
- You cannot save session layouts. If you close a window, you can restore the original layout using **File** > **Restore Original View**.

Related Examples

- “Standalone Boot Method”

More About

- “Guidelines for Exporting Explorer Configuration” on page 4-12

Guidelines for Exporting Explorer Configuration

Before exporting a Simulink Real-Time Explorer configuration, set up the execution environment, signal and parameter groups, and panels. Lay out the windows the way that you want them in the standalone executable. Follow these guidelines:

Execution Environment

For each computer on which you intend to run the standalone Simulink Real-Time Explorer executable:

- Verify that the computer is compatible with Windows and that the CPU and operating system meet the requirements for executing the standalone Simulink Real-Time Explorer executable.
- Verify that Microsoft .NET Framework 4.5 is installed on the computer.
- Record whether the computer has a 64-bit or a 32-bit architecture. You need this information to run the export program.

For each target computer on which you intend to run the real-time application:

- 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 to connect each computer that is compatible with Windows to each target computer.

  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 `TargetPC`*x* to something more specific to your system.

- 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 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.

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.

Instrument Panels

- To interact with multiple target computers, create a separate instrument panel for each separate real-time application and 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"

More About

- “Explorer Configuration Exported to Run Outside MATLAB” on page 4-10

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 .

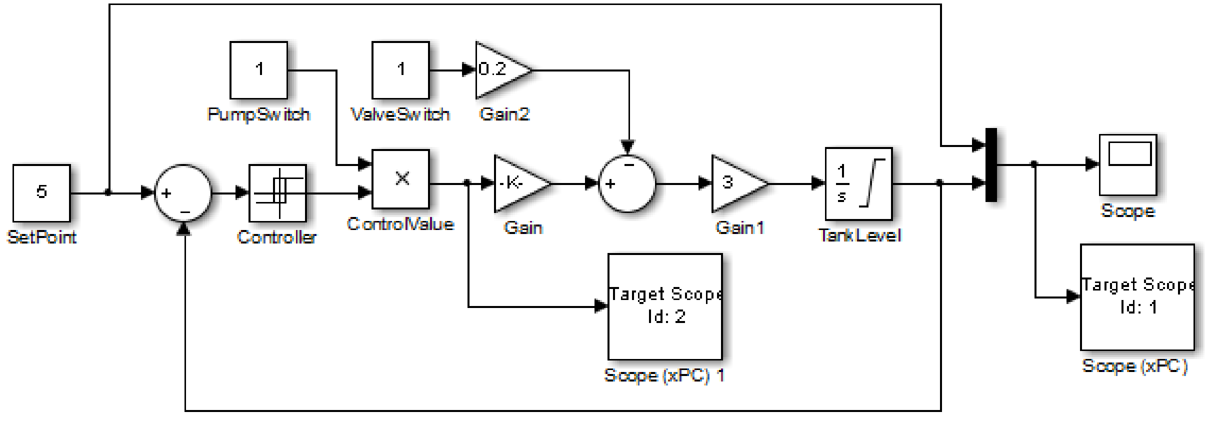
The next task is “Configure Instrument for Parameter Tuning” on page 4-16.

Related Examples

- “Save and Restore Layouts” on page 4-24

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” on page 4-15).



Tank Level Control System

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 to 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). 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.

Click 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`.

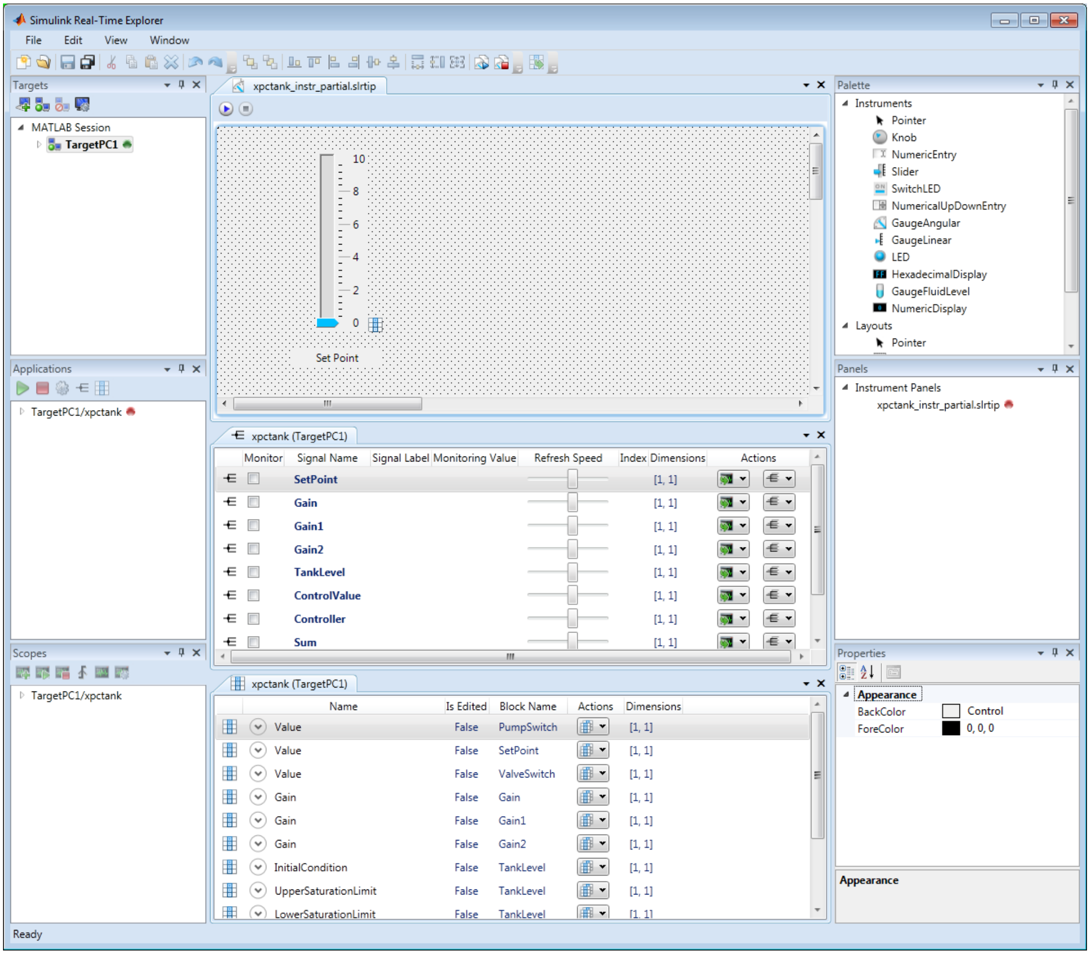
9 Scroll down to the **TextAlign** property. Click the down arrow and click the center of the nine blocks presented.

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" on page 4-19.

Related Examples

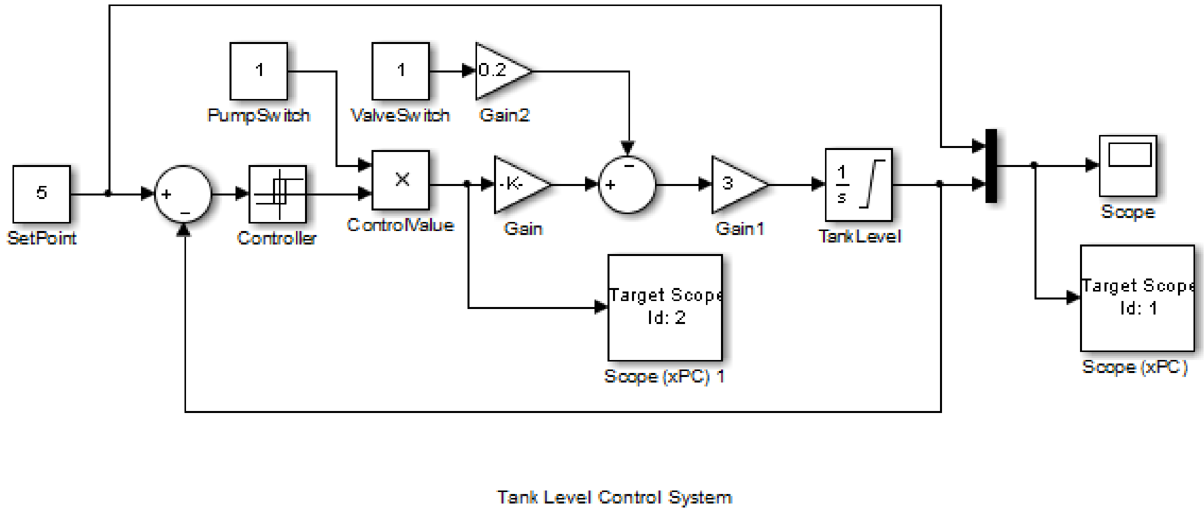
- "Save and Restore Layouts" on page 4-24

More About

- "Instrumentation for Real-Time Applications" on page 4-2

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” on page 4-15).



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 to 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). 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 click the center of the nine blocks presented.

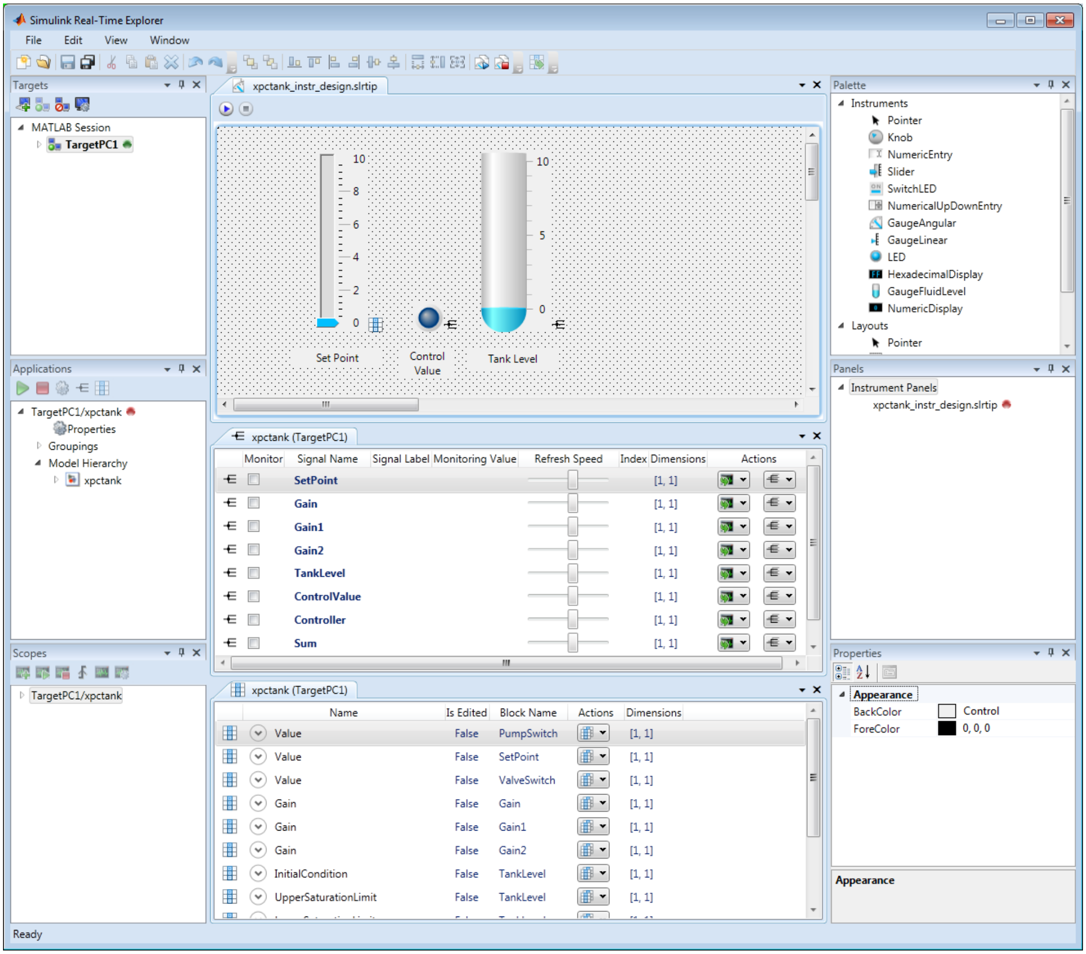
The **TextAlign** property becomes `MiddleCenter`.

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” on page 4-25.

Related Examples

- “Save and Restore Layouts” on page 4-24

More About

- "Instrumentation for Real-Time Applications" on page 4-2

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” on page 4-16 and “Configure Instruments for Signal Display” on page 4-19.

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`.

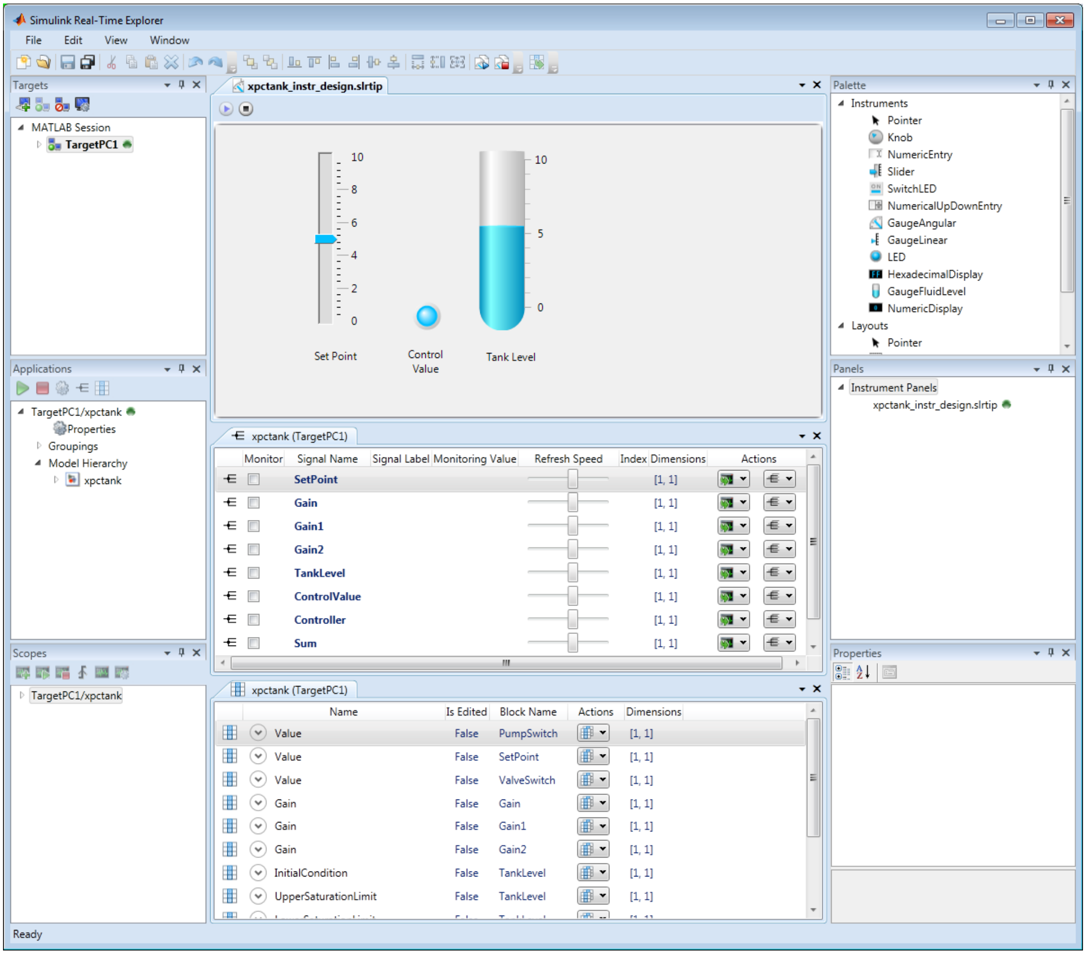
2 Set property **Stop time** to `inf` in the **Applications** pane (on the toolbar).

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 on the toolbar.

7 To stop the instruments, in the `xpctank_instr_design.slrtip` instrument panel, click the Stop Instrument icon.

Related Examples

- "Real-Time Application Instruments"

More About

- "Instrumentation for Real-Time Applications" on page 4-2

Prepare Explorer Environment for Export

Verify that each combination of a computer that is compatible with Windows and a 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 “Real-Time Application Instruments”.

For each computer on which you intend to run the standalone Simulink Real-Time Explorer executable:

1. Verify that the computer is compatible with Windows and that the CPU and operating system meet the requirements for executing the standalone Simulink Real-Time Explorer executable.
2. Verify that Microsoft .NET Framework 4.5 is installed on the computer.
3. Record whether the computer has a 64-bit or a 32-bit architecture. You need this information to run the export program.

For each target computer on which you intend to run the real-time application:

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.

   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. In the **Boot configuration** tab, set **Boot mode** to `Stand Alone`.
6. Connect the target computer to the computer compatible with Windows and restart the target computer. Verify that the target computer loads the Simulink Real-Time kernel and starts the real-time application.

Related Examples

- “Development Computer Requirements”
- “Target Computer Requirements”
- “Ethernet Link Setup”
- “Target Computer Settings”
- “Standalone Boot Method”

More About

- “Explorer Configuration Exported to Run Outside MATLAB” on page 4-10
- “Guidelines for Exporting Explorer Configuration” on page 4-12

Prepare Instrument Panel Configuration for Export

Load the instrument panels for the instrumented model. Resize and lay out the Simulink Real-Time Explorer windows.

The example uses the instrumented `xpctank` model.

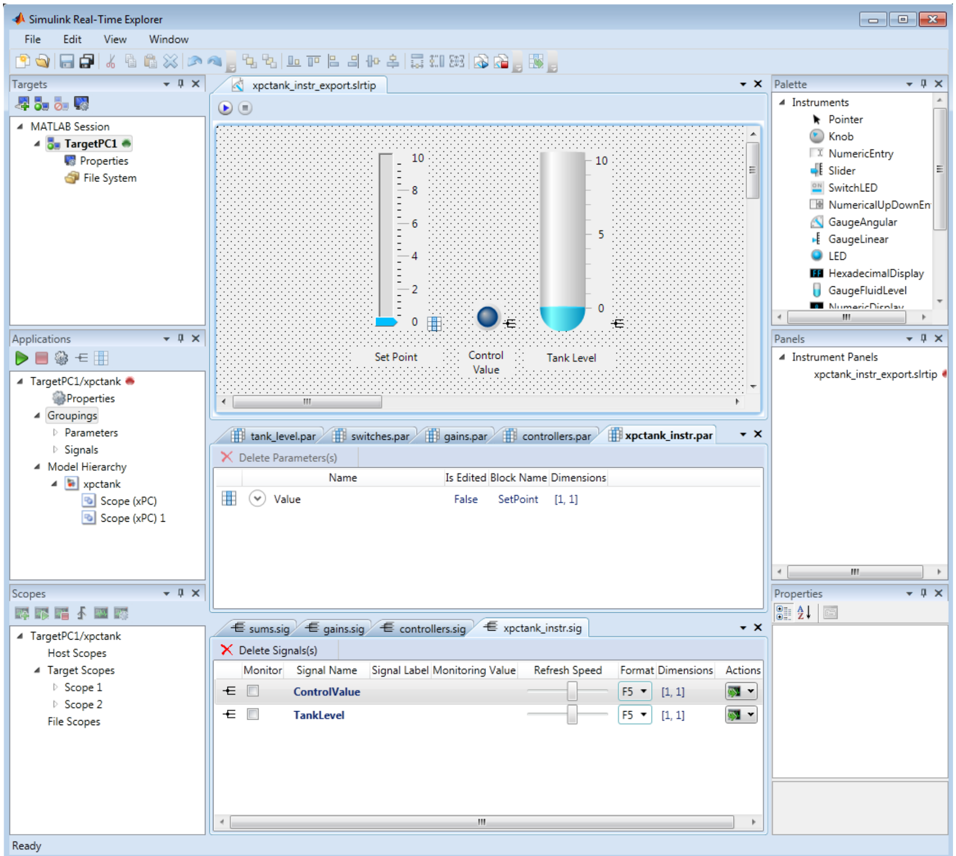
Note: When you run the standalone executable, you cannot access the model hierarchy. You can access only instrument panels and windows that were open when you exported the configuration. You can access only signals and parameters that were loaded in signal and parameter groups when you exported the configuration.

---

1. Load your instrument panels into Simulink Real-Time Explorer.

   Here, the panel is `xpctank_instr_design.slrtip`.
2. Create a parameter group for the key block representing the set point (`xpctank_instr.par`).
3. Create parameter groups for the low-level blocks representing the tank level, switches, gains, and controllers (`tank_level.par`, `switches.par`, `gains.par`, and `controllers.par`).
4. Create a signal group for the key blocks representing the control value and tank level (`xpctank_instr.sig`).
5. Create signal groups for the low-level blocks representing the sums, gains, and controllers (`sums.sig`, `gains.sig`, and `controllers.sig`).
6. Open, lay out, and resize the windows that you want the standalone executable to open.
7. Save each instrument panel.

The `xpctank_instr_design.slrtip` configuration looks like the figure.



The next task is “Export Explorer Configuration” on page 4-32.

Related Examples

- “Create Parameter Groups Using Simulink Real-Time Explorer”
- “Create Signal Groups Using Simulink Real-Time Explorer”

More About

- “Instrumentation for Real-Time Applications” on page 4-2

Export Explorer Configuration

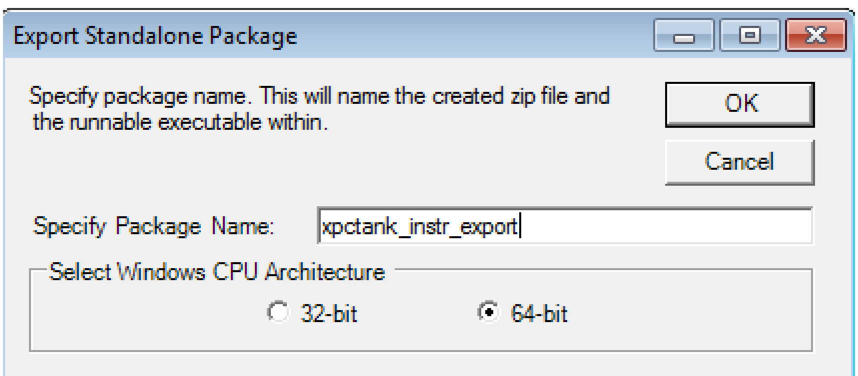
Export the Simulink Real-Time Explorer configuration as a standalone executable.

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” on page 4-30.

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 loaded in 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**.



4. Click **OK**.

   The software generates a file named `xpctank_instr_export.zip` in the current folder.

The next task is “Unpack and Run Standalone Configuration” on page 4-34.

More About

- "Instrumentation for Real-Time Applications" on page 4-2

Unpack and Run Standalone Configuration

Unpack the standalone executable onto the computer that is compatible with Windows.

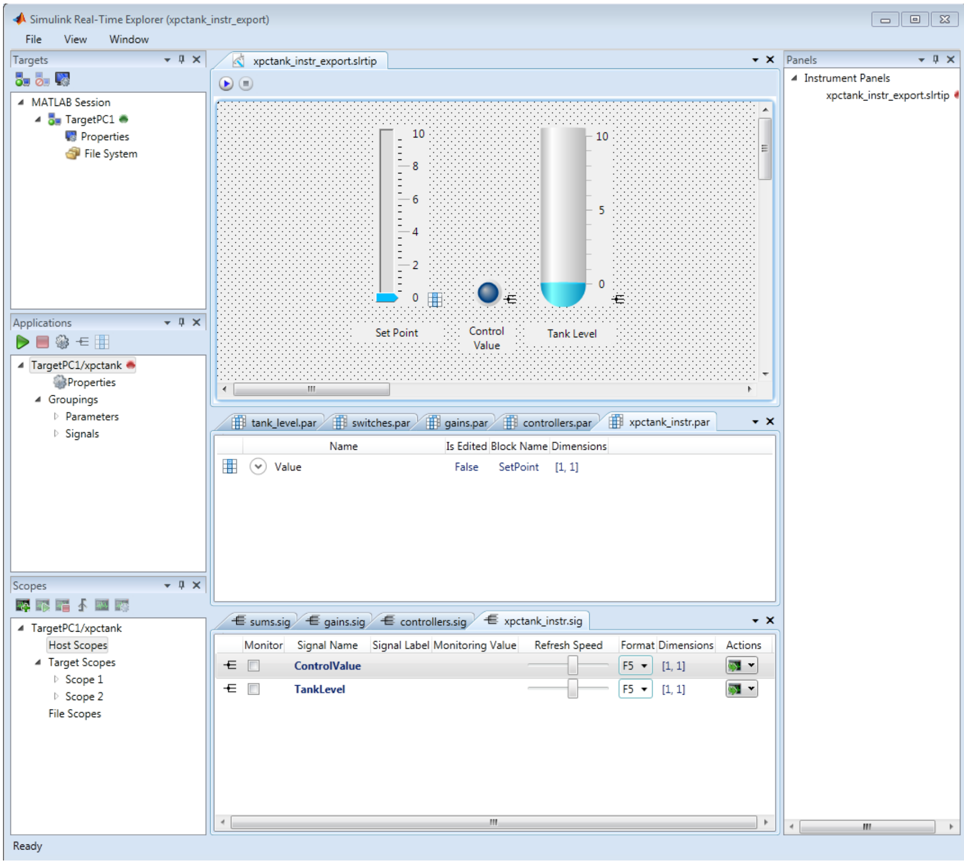
The example uses the instrumented `xpctank` model. Before carrying out this procedure, you must have performed the steps in "Export Explorer Configuration" on page 4-32.

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 loaded in signal and parameter groups when you exported the configuration.

---

1. Copy `xpctank_instr_export.zip` from the original folder to a folder on the computer compatible with Windows, 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 to folder `xpctank_instr_export` in the extraction root folder.
6. Connect the target computer to the computer that is compatible with Windows. Restart the target computer.

   Verify that the target computer loads the Simulink Real-Time kernel and the real-time application.
7. Double-click `runxpctank_instr_export.exe`.



To interact with the real-time application on the target computer, use the executable interface.

- 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

- "Instrumentation for Real-Time Applications" on page 4-2

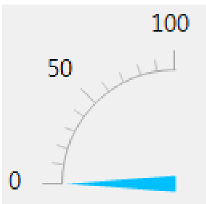
Instruments — Alphabetical List

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

DesiredIncrement	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.
FixedMinMaxMajor	If `True`, the top and bottom ticks are constrained to be major ticks with min/max values defined by **Min** and **Span**
MidIncluded	If `True`, insert a tick halfway between major ticks. If **MinorCount** is even, space the minor ticks equally around the center tick. If **MinorCount** is odd, replace the center tick with the middle tick. If
MinorCount	Number of minor ticks between major ticks
MinTextSpacing	Minimum space between scale ticks

Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	Display unit next to tick labels

General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
Min	Minimum possible value
Reverse	If `True`, flip the display to increase in the opposite direction
ScaleType	One of the values `Linear`, `Log10`, and `SplitLinearLog10`
Span	Number of values between the minimum and maximum values

Angular Scale Range

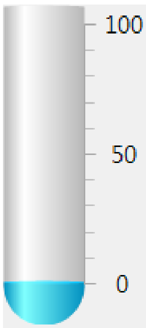
The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
AngleMin	Specify starting point of scale, from bottom of circle
AngleSpan	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
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
DesiredIncrement	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.
FixedMinMaxMajor	If `True`, the top and bottom ticks are constrained to be major ticks with min/max values defined by **Min** and **Span**
MidIncluded	If `True`, insert a tick halfway between major ticks. If **MinorCount** is even, space the minor ticks equally around the center tick. If **MinorCount** is odd, replace the center tick with the middle tick. If
MinorCount	Number of minor ticks between major ticks
MinTextSpacing	Minimum space between scale ticks

Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	Display unit next to tick labels

General Scale Range

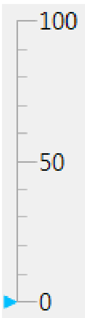
The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
Min	Minimum possible value
Reverse	If `True`, flip the display to increase in the opposite direction
ScaleType	One of the values `Linear`, `Log10`, and `SplitLinearLog10`
Span	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
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
DesiredIncrement	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.
FixedMinMaxMajor	If `True`, the top and bottom ticks are constrained to be major ticks with min/max values defined by **Min** and **Span**
MidIncluded	If `True`, insert a tick halfway between major ticks. If **MinorCount** is even, space the minor ticks equally around the center tick. If **MinorCount** is odd, replace the center tick with the middle tick. If
MinorCount	Number of minor ticks between major ticks
MinTextSpacing	Minimum space between scale ticks

Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	Display unit next to tick labels

General Scale Range

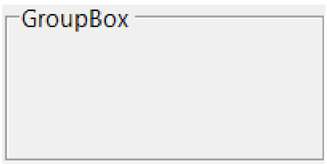
The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
Min	Minimum possible value
Reverse	If `True`, flip the display to increase in the opposite direction
ScaleType	One of the values `Linear`, `Log10`, and `SplitLinearLog10`
Span	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
AutoSize	If `True`, the box expands at design time to make visible the instruments within it
AutoSizeMode	Possible values are `GrowAndShrink` and `GrowOnly`. The default is `GrowOnly`.

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.

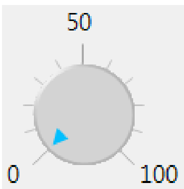
To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots to the right of the group.

Parameter	Usage
AutoSize	If `True`, the box expands at design time to make visible the specified digits. The default is `True`.
DigitCount	Number of hex digits to be displayed
DigitLeading	Possible values are `None` and `Zeros`.

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
Enabled	If `True`, the switch is visible
On	If `True`, the switch is on

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
DesiredIncrement	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.
FixedMinMaxMajor	If `True`, the top and bottom ticks are constrained to be major ticks with min/max values defined by **Min** and **Span**
MidIncluded	If `True`, insert a tick halfway between major ticks. If **MinorCount** is even, space the minor ticks equally around the center tick. If **MinorCount** is odd, replace the center tick with the middle tick. If
MinorCount	Number of minor ticks between major ticks
MinTextSpacing	Minimum space between scale ticks

Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`

UnitsText	Display unit next to tick labels

General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
Min	Minimum possible value
Reverse	If `True`, flip the display to increase in the opposite direction
ScaleType	One of the values `Linear`, `Log10`, and `SplitLinearLog10`
Span	Number of values between the minimum and maximum values

Angular Scale Range

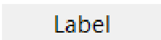
The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
AngleMin	Specify starting point of scale, from bottom of circle
AngleSpan	Specify number of degrees taken up by scale

Label

Graphic container for text

Description



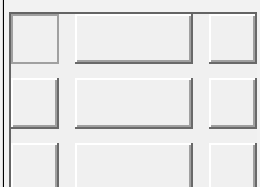
Use the **Label** graphic to add text to the instrument layout.

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
Text	Contains the text displayed by the label
TextAlign	Specifies left-right, top-bottom alignment using a `3x3` matrix. This display represents setting `TopLeft`. 

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
AutoSize	If `True`, size the graphic to accommodate the specified graphic parameters.
BlinkerEnable	If `True`, **LED** graphic blinks continuously.

Indicator Parameters

The root node of these parameters is **Instrument+Indicator**.

Parameter	Usage
ColorActive	Indicator color if signal value is 1.

Parameter	Usage
ColorInactive	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 specified 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
AutoSize	If `True`, the box expands at design time to make visible the specified digits. The default is `True`.

Value Display

The root node of these parameters is **Iocomp+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point

PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	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 specified 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**
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	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
AutoSize	If `True`, the box expands at design time to make visible the specified digits. The default is `False`.

Scale Range

The root node of these parameters is **Data**.

Parameter	Usage
DecimalPlaces	Number of decimal places to display

Increment	Value to add or subtract in response to an up-arrow or down-arrow
Maximum	Maximum data value
Minimum	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
AutoScroll	If `True`, the box scrolls at run time to make fully visible partially visible instruments within it.
AutoSize	If `True`, the box expands at design time to make visible the instruments within it.
AutoSizeMode	Possible values are `GrowAndShrink` and `GrowOnly`. The default is `GrowOnly`

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.

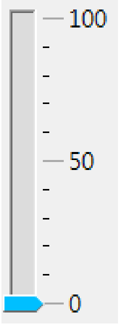
To access a parameter dialog box for the instrument as a whole, select the instrument and click the Tasks icon in the top right corner. To access a dialog box for a parameter group, click the group, and then click the continuation dots to the right of the group.

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
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
DesiredIncrement	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.
FixedMinMaxMajor	If `True`, the top and bottom ticks are constrained to be major ticks with min/max values defined by **Min** and **Span**
MidIncluded	If `True`, insert a tick halfway between major ticks. If **MinorCount** is even, space the minor ticks equally around the center tick. If **MinorCount** is odd, replace the center tick with the middle tick. If
MinorCount	Number of minor ticks between major ticks
MinTextSpacing	Minimum space between scale ticks

Scale Text Display

The root node of these parameters is **Instrument+ScaleDisplay+TextFormatting**.

Parameter	Usage
Precision	Number of digits to the right of the decimal point
PrecisionStyle	One of the values `FixedDecimalPoints`, `SignificantDigits`, `None`
Style	One of the values `Number`, `Thousands`, `Prefix`, `Exponent`, `Price32nds`, `DateTime`, `DateTimeUTC`
UnitsText	Display unit next to tick labels

General Scale Range

The root node of these parameters is **Instrument+ScaleRange**.

Parameter	Usage
Min	Minimum possible value
Reverse	If `True`, flip the display to increase in the opposite direction
ScaleType	One of the values `Linear`, `Log10`, and `SplitLinearLog10`
Span	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
ColorInactive	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 Real-Time Application at Target Computer Command Line” for a description of how to use these functions and commands.

In this section...

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
`start`	Start the real-time application currently loaded on the target computer.	`start(tg)`

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`
 	
`stoptime = Inf`
	With the value `number`, run for the specified number of seconds.
	
	With the value `Inf`, run the real-time application until you manually stop it or reset the target computer.
		`tg.StopTime = number`
		
		`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

`addscope`
 	
`addscope scope_index`
	Without an argument, add a target scope and assign it the next available index.
	
	With argument `scope_index`, add a target scope and assign it index `scope_index`.
		`addscope(tg, 'target')`
		
		`addscope(tg, 'target', scope_index)`

`remscope scope_index`
 	
`remscope all`
	With value `scope_index`, remove scope `scope_index`.
	
	With value `all`, remove all scopes.
		`remscope(tg, scope_index)`
		
		`remscope(tg)`

Target Computer Command	Description	MATLAB Equivalent

`startscope scope_index`
 	
`startscope all`
	With value `scope_index`, start the scope with index `scope_index`.
	
	With value `all`, start all scopes.
		`start(sc)`
		
		`start(getscope(tg))`

`stopscope scope_index`
 	
`stopscope all`
	With value `scope_index`, stop the scope with index `scope_index`.
	
	With value `all`, stop all scopes.
		`stop(sc)`
		
		`stop(getscope(tg))`

`addsignal scope_index = sig_index1, sig_index2, ...`	With values `sig_index1`, `sig_index2`, `...`, add the signals with these signal indexes to the scope with index `scope_index`.	`addsignal(sc, sig_index_vector)`

`remsignal scope_index = sig_index1, sig_index2, ...`
 	
`remsignal scope_index`
	With values `sig_index1`, `sig_index2`, `...`, remove the signals with these signal indexes from the scope with index `scope_index`.
	
	Without a `sig_index` value, remove all the signals from the scope with index `scope_index`.
		`remsignal(sc, sig_index_vector)`
		
		`remsignal(sc)`

`show Scope scope_index`	With value `scope_index`, expand scope `scope_index` from an icon.	
`hide Scope scope_index`	With value `scope_index`, collapse scope `scope_index` into an icon.	
`show Video video_index`	With value `video_index`, expand video display `video_index` 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 decimation scope_index = number	With value 1, the scope returns all sample points. With value n, the scope returns every nth sample point.	sc.Decimation = 1 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`
 	
`scopemode scope_index = 1`
 	
`scopemode scope_index = redraw`
 	
`scopemode scope_index = 3`
 	
`scopemode scope_index = rolling`
	displays signal values as text.
	
	With value `1` or `redraw`, scope `scope_index` plots signal values when `numsamples` samples has been acquired.
	
	With value `3` or `rolling`, scope `scope_index` plots signal values at every sample time.
	
	Note: Value `2`, `sliding`, will be removed in a future release. It behaves like value `3`, `rolling`.
		`sc.DisplayMode = 'redraw'`
		
		`sc.DisplayMode = 'rolling'`

Target Computer Command	Description	MATLAB Equivalent

`triggermode scope_index = 0`
 	
`triggermode scope_index = freerun`
 	
`triggermode scope_index = 1`
 	
`triggermode scope_index = software`
 	
`triggermode scope_index = 2`
 	
`triggermode scope_index = signal`
 	
`triggermode scope_index = 3`
 	
`triggermode scope_index = scope`
	With value `0` or `freerun`, scope `scope_index` triggers on every sample time.
	
	With value `1` or `software`, scope `scope_index` triggers from Command Window.
	
	With value `2` or `signal`, scope `scope_index` triggers when a designated signal changes state.
	
	With value `3` or `scope`, scope `scope_index` triggers when a designated scope triggers.
		`sc.TriggerMode = 'freerun'`
		
		`sc.TriggerMode = 'software'`
		
		`sc.TriggerMode = 'signal'`
		
		`sc.TriggerMode = '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`
 	
`triggerslope scope_index = either`
 	
`triggerslope scope_index = 1`
 	
`triggerslope scope_index = rising`
 	
`triggerslope scope_index = 2`
 	
`triggerslope scope_index = falling`
	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'`
		
		`sc.TriggerSlope = 'Rising'`
		
		`sc.TriggerSlope = '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

`ylimit scope_index = min_y, max_y`
 	
`ylimit scope_index = auto`
	With value `min_y, max_y`, change the lower and upper *y*-axis values to `min_y` and `max_y`.
	
	With value `auto`, allow the lower and upper *y*-axis values to be determined by the values being displayed.
		`sc.YLimit = [min_y, max_y]`
		
		`sc.YLimit = 'auto'`

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
`setvar variable_name = command`	Set a variable to a target computer command line string.
`getvar variable_name`	Display the value of a variable.
`delvar variable_name`	Delete a variable.
`delallvar`	Delete all variables.
`showvar`	Display a list of variables.

6

Simulink Real-Time Performance Advisor Checks

Simulink Real-Time Performance Advisor Checks

In this section...

Performance Advisor Check Overview

Use Performance Advisor checks to improve real-tune application execution time. Performance Advisor runs the check and only provides recommendations. It does not modify your model.

See Also

"How Performance Advisor Improves Simulation Performance"

Baseline

Checks preconditions for real-time execution, and then performs an initial measurement to collect baseline performance data. Performance Advisor later uses this baseline to compare performance before and after you implement the improvements that Performance Advisor recommends.

See Also

- "Performance Optimization"

- “How Performance Advisor Improves Simulation Performance”

System Target File Compatibility

Checks that the system target file is compatible with the real-time advisor workflow.

In the Configuration Parameters **Code Generation** pane, set the **System target file** setting to either `slrt.tlc` or `slrtert.tlc`.

See Also

- “Set Configuration Parameters”
- “Performance Optimization”
- “How Performance Advisor Improves Simulation Performance”

Profiling Settings

Checks that the execution profiling settings are compatible with the real-time advisor workflow.

In the Configuration Parameters **Verification** pane, select the **Measure task execution time** check box.

See Also

- “Configure Real-Time Application for Profiling”
- “Performance Optimization”
- “How Performance Advisor Improves Simulation Performance”

Check Target

Pings the target computer and verifies that it is in a clean state.

To clear the target computer of prior simulations, select the **Automatically unload any loaded application** check box.

See Also

- “Performance Optimization”

- “How Performance Advisor Improves Simulation Performance”

Real-Time Performance Baseline

Collects execution-time measurements and establishes a performance baseline.

This check builds, downloads, and executes the real-time application on the target computer. When the check passes, it displays the following information:

- **Margin before CPU overload (0% indicates CPU overload)** — A table containing the real-time application task name, the sample rate, and the margin.

  Margin is the minimum value of headroom for a task over all the measured samples.

  Headroom is the time between the end of execution and the start of the next sample, as a percentage of sample time. For example, if the sample time is `1.2` ms and a task takes `0.93` ms to execute, the headroom is `(1.2 - 0.93)/1.2`, or `22.5%`.

  As the margin approaches 0%, the application gets closer to overloading the CPU.
- **Average CPU Usage** — A pie chart showing the average CPU resources that the real-time application uses, as a percentage of available resources.

  The available CPU resources include all of the processors on a multicore target computer. For example, a single-tasking model running on a quad-core processor cannot exceed `25%` CPU usage.

  The background task aggregates the CPU time for all operating system tasks that are not related to application execution. These tasks include updating the target screen, communicating with the development computer, and so on. It also includes the time that the CPU is idle.

See Also

- “Performance Optimization”
- “How Performance Advisor Improves Simulation Performance”

Determine minimum sample time

To determine the minimum sample time possible for the model, this check iteratively runs the model, decreasing the sample time on each run. The algorithm stops at a sample time that is greater than a sample time that can cause an overload.

Random factors such as network latency can change the minimum sample time of a model. As a best practice, set the sample time for your model to a value greater than the minimum sample time returned by this check.

This check builds, downloads, and executes the real-time application.

See Also

- "Performance Optimization"
- "How Performance Advisor Improves Simulation Performance"

Real-Time

Checks the real-time application and application setup, and recommends changes to improve performance.

See Also

- "Performance Optimization"
- "How Performance Advisor Improves Simulation Performance"

Outport Logging

Checks for the use of Outport blocks for data logging. Data logging using Outport blocks can slow down execution.

As an alternative, consider using a real-time Scope block configured as a file scope.

See Also

- `Scope`
- "Performance Optimization"
- "How Performance Advisor Improves Simulation Performance"

EtherCAT Synchronous SDO

Checks for the use of EtherCAT Synchronous SDO blocks. EtherCAT Synchronous SDO blocks can slow down execution.

As an alternative, consider using EtherCAT Asynchronous SDO blocks.

See Also

- “EtherCAT”
- “Performance Optimization”
- “How Performance Advisor Improves Simulation Performance”

Concurrent execution

Checks if you can perform concurrent execution of the real-time application on a multicore target computer.

To perform concurrent execution:

- Choose a multicore target computer.
- In the Simulink Real-Time Explorer **Target settings** pane, select the **Multicore CPU** check box.
- In the Configuration Parameters **Solver** pane, select the **Allow tasks to execute concurrently on target** check box.

This check updates the model diagram.

See Also

- “Concurrent Execution Using Multicore Target Computer”
- “Performance Optimization”
- “How Performance Advisor Improves Simulation Performance”

Final Validation

Validates the overall performance improvement that your changes make in real-time execution time and accuracy.

If you have not validated the performance improvement resulting from other checks, use this check to perform a final validation of the changes to the model.

This check builds, downloads, and executes the real-time application. When the check passes, it displays the following information for the baseline and final validation runs:

- **Margin before CPU overload (0% indicates CPU overload)** — A table containing, for each run, the real-time application task name, the sample rate, and the margin.

  Margin is the minimum value of headroom for a task over all the measured samples.

  Headroom is the time between the end of execution and the start of the next sample, as a percentage of sample time. For example, if the sample time is `1.2` ms and a task takes `0.93` ms to execute, the headroom is `(1.2 - 0.93)/1.2`, or `22.5%`.

  As the margin approaches 0%, the application gets closer to overloading the CPU.
- **Average CPU Usage** — Pie charts showing, for each run, the average CPU resources that the real-time application uses, as a percentage of available resources.

  The available CPU resources include all of the processors on a multicore target computer. For example, a single-tasking model running on a quad-core processor cannot exceed `25%` CPU usage.

  The background task aggregates the CPU time for all operating system tasks that are not related to application execution. These tasks include updating the target screen, communicating with the development computer, and so on. It also includes the time that the CPU is idle.

See Also

- "Performance Optimization"
- "How Performance Advisor Improves Simulation Performance"
- "Comparing Performance"