Simulink® Real-Time™

API Guide

MATLAB®&SIMULINK®

R2020b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Simulink® Real-Time™ API Guide

Trademarks

Patents

Revision History

July 2002	Online only	New for Version 2 (Release 13)
October 2002	Online only	Updated for Version 2 (Release 13)
September 2003	Online only	Revised for Version 2.0.1 (Release 13SP1)
June 2004	Online only	Revised for Version 2.5 (Release 14)
August 2004	Online only	Revised for Version 2.6 (Release 14+)
October 2004	Online only	Revised for Version 2.6.1 (Release 14SP1)
November 2004	Online only	Revised for Version 2.7 (Release 14SP1+)
March 2005	Online only	Revised for Version 2.7.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.8 (Release 14SP3)
March 2006	Online only	Revised for Version 2.9 (Release 2006a)
May 2006	Online only	Revised for Version 3.0 (Release 2006a+)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.4 (Release 2008a)
October 2008	Online only	Revised for Version 4.0 (Release 2008b)
March 2009	Online only	Revised for Version 4.1 (Release 2009a)
September 2009	Online only	Revised for Version 4.2 (Release 2009b)
March 2010	Online only	Revised for Version 4.3 (Release 2010a)
September 2010	Online only	Revised for Version 4.4 (Release 2010b)
April 2011	Online only	Revised for Version 5.0 (Release 2011a)
September 2011	Online only	Revised for Version 5.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.2 (Release 2012a)
September 2012	Online only	Revised for Version 5.3 (Release 2012b)
March 2013	Online only	Revised for Version 5.4 (Release 2013a)
September 2013	Online only	Revised for Version 5.5 (Release 2013b)
March 2014	Online only	Revised for Version 6.0 (Release 2014a)
October 2014	Online only	Revised for Version 6.1 (Release 2014b)
March 2015	Online only	Revised for Version 6.2 (Release 2015a)
September 2015	Online only	Revised for Version 6.3 (Release 2015b)
March 2016	Online only	Revised for Version 6.4 (Release 2016a)
September 2016	Online only	Revised for Version 6.5 (Release 2016b)
March 2017	Online only	Revised for Version 6.6 (Release 2017a)
September 2017	Online only	Revised for Version 6.7 (Release 2017b)
March 2018	Online only	Revised for Version 6.8 (Release 2018a)
September 2018	Online only	Revised for Version 6.9 (Release 2018b)
March 2019	Online only	Revised for Version 6.10 (Release 2019a)
September 2019	Online only	Revised for Version 6.11 (Release 2019b)
March 2020	Online only	Revised for Version 6.12 (Release 2020a)
September 2020	Online only	Revised for Version 7.0 (Release 2020b)

Contents

MATLAB API

slrtExplorer

Open Simulink Real-Time explorer and interact with target computers and real-time applications

Syntax

```
slrtExplorer
```

Description

`slrtExplorer` opens the Simulink Real-Time explorer.

Simulink Real-Time explorer provides a UI for viewing connection status and interacting with a real-time application. You can:

- View a hierarchical display of signals.
- Tune parameters.
- Stream data to the Simulation Data Inspector.

Examples

Select Signals and Stream Data

The explorer provides a view of signals in the real-time application. From this view, you can select signals to stream to the Simulation Data Inspector and visualize the data..

Open the Simulink Real-Time explorer. Type:

```
slrtExplorer
```

To connect to the selected target computer, click **Connect**.

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

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

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

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

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

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

See Also

`slrtLogViewer` | `slrtTETMonitor`

Topics

Simulink Real-Time Explorer

Introduced in R2020b

slrtLogViewer

Open Simulink Real-Time System Log Viewer tab in Simulink Real-Time Explorer to view the console log from target computer

Syntax

```
slrtLogViewer
```

Description

`slrtLogViewer` opens Simulink Real-Time Explorer and shows the System Log Viewer tab.

Examples

Open System Log Viewer

Open Simulink Real-Time Explorer and show the System Log Viewer tab.

```
slrtLogViewer
```

See Also

`SystemLog` | `slrtExplorer` | `slrtTETMonitor`

Topics

Simulink Real-Time Explorer

Introduced in R2020b

slrtTETMonitor

Open Simulink Real-Time task execution time (TET) monitor

Syntax

```
slrtTETMonitor
```

Description

`slrtTETMonitor` opens the Simulink Real-Time task execution time (TET) monitor in the MATLAB session that is available for all Simulink Real-Time target objects. You can open the TET monitor at any time. Depending on the current state of connected target computers, the monitor displays TET data for each real-time application task. Changes to the target computer state are updated in the TET monitor. The monitor displays these target states:

- *`target_name`* **Waiting for real-time execution to start**: Displays name of target computer connected to Simulink Real-Time. Displays no TET data is because no real-time application is loaded or executing.
- *`target_name`* **BaseRate** *`rate_value`*: Displays TET data for execution of the real-time because a real-time application is executing.

Examples

Open TET Monitor and View Status

In the “Data Logging with Simulation Data Inspector (SDI)” example, use these additional steps to display the TET monitor.

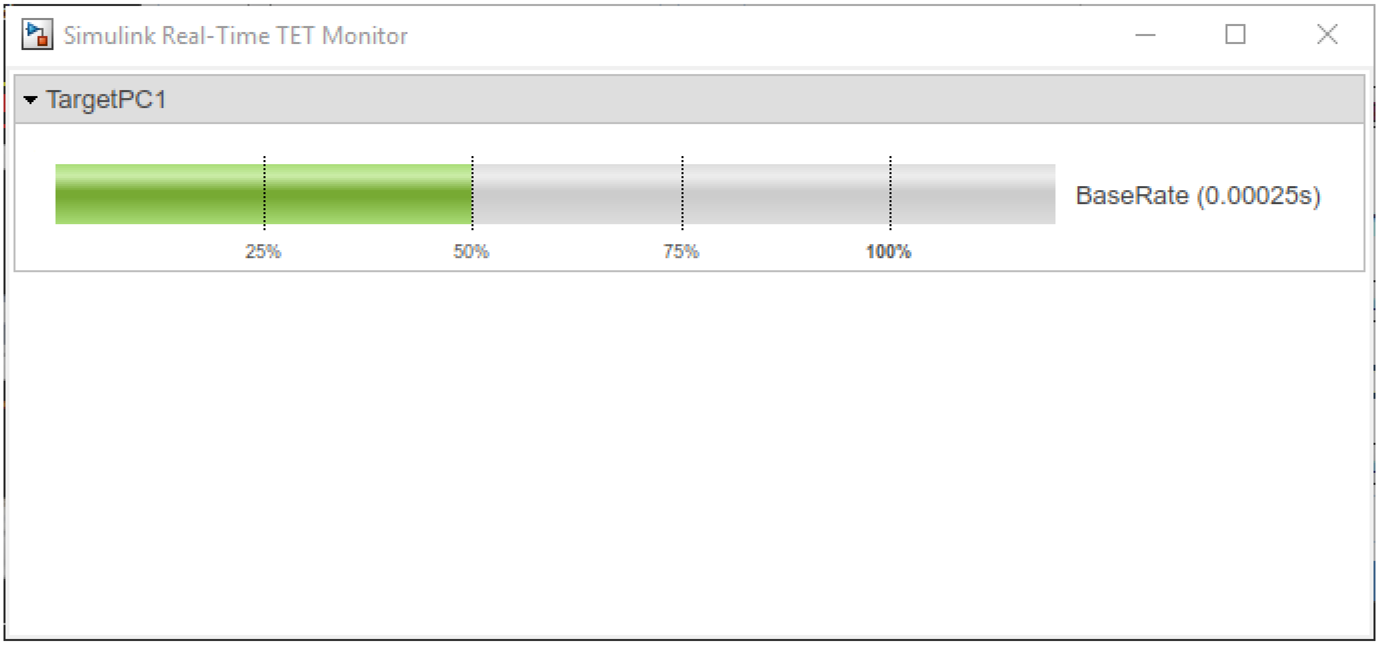
Open the `slrt_ex_osc` model.

Build the real-time application, load it on the target computer, and start the application. In Simulink Editor **Real-Time** tab, click **Run on Target**.

Open the TET monitor. In the **Real-Time** tab, click **TET Monitor**. Or, in the Command Window, enter:

```
slrtTETMonitor
```

When you run the real-time application, the TET monitor displays status.



View TET Data in Simulation Data Inspector

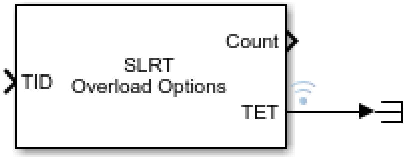
In the “Data Logging with Simulation Data Inspector (SDI)” example, use these additional steps to display the TET data in the Simulation Data Inspector.

Open the `slrt_ex_osc` model.

Add a SLRT Overload Options block to the model.

In the block, set the **Enable TET Output** parameter value to `on`.

Select the TET output and mark it for data logging in the Simulation Data Inspector.



Build the real-time application, load it on the target computer, and start the application. In Simulink Editor **Real-Time** tab, click **Run on Target**.

Open the Simulation Data Inspector.

When you run the real-time application, the TET data is displayed in the Simulation Data Inspector.

See Also

SLRT Overload Options | **Simulink Real-Time TET Monitor** | `slrtExplorer` | `slrtLogViewer`

Topics

“Data Logging with Simulation Data Inspector (SDI)”
Simulink Real-Time Explorer

Introduced in R2020b

Target

Represent real-time application and target computer status

Description

A `Target` object represents a target computer and provides access to methods and properties related to the target computer.

The object provides access to methods and properties that:

- Start and stop the real-time application.
- Read and set parameters.
- Monitor signals.
- Retrieve status information about the target computer.
- Restart the target computer.
- Load the real-time application.
- Start, stop, and retrieve information from the profiler.

Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name if the characters you type are unique for the property.

You can invoke some of the object properties and functions from the target computer command line when the real-time application has been loaded. For more information, see “Target Computer Command-Line Interface”.

Creation

`target_object = slrealtime` constructs a target object representing the default target computer.

`target_object = slrealtime(target_name)` constructs a target object representing the target computer designated by `target_name`.

The `slrealtime` function accepts these arguments:

- `target_name` — Name assigned to target computer (character vector or string scalar). For example, `'TargetPC1'`.
- `target_object` — Object representing target computer. For example, `tg`.

Example: “Create Target Object for Default Target Computer” on page 1-11

Example: “Build and Run Real-Time Application” on page 1-12

Target Object Properties

TargetSettings — Target computer configuration information

TargetSettings struct

The TargetSettings property holds a TargetSettings structure that includes fields name, address, sshPort, xcpPort, username, userPassword, and rootPassword. To view the targetSettings, in the MATLAB Command Window, type

```
tg.TargetSettings
```

```
ans = 

  TargetSettings with properties:

            name: 'TargetPC1'
         address: '10.10.10.35'
         sshPort: 22
         xcpPort: 5555
        username: 'slrt'
    userPassword: 'slrt'
    rootPassword: 'root'
```

ProfilerStatus — Target computer execution profiler information

Ready | StartRequested | Running | DataAvailable

The ProfilerStatus property holds the execution profiler status. To view the ProfilerStatus, in the MATLAB Command Window, type

```
tg.ProfilerStatus
```

```
ans = 

    'Ready'
```

SDIRunId — Target computer SDI run identifier

int32

The SDIRunId property holds the Simulation Data Inspector run identifier for the current simulation run. To view the SDIRunId, in the MATLAB Command Window, type

```
tg.SDIRunId
```

```
ans = 

  int32

   22110
```

ptpd — Target computer PTP daemon configuration

PTPControl struct

The ptpd property holds a PTPControl structure that includes fields Command and AutoStart. For more information, see the Target.ptpd object. To view the targetSettings, in the MATLAB Command Window, type

```
tg.ptpd
```

```
ans = 

  PTPControl with properties:

      Command: 'ptpd -L -K -g'
    AutoStart: 1
```

FileLog — Target computer file logger status information

`FileLogger` struct

The `FileLog` property holds a `FileLogger` structure that includes fields `Importing`, `ImportProgress`, `LoggingService`, and `DataAvailable`. For more information, see the `Target.FileLog` object. To view the `targetSettings`, in the MATLAB Command Window, type

```
tg.FileLog
```

```
ans = 

  FileLogger with properties:

         Importing: 0
    ImportProgress: 100
    LoggingService: STARTING
     DataAvailable: 0
```

Events

A number of the `Target` object functions produce event status. You can use the MATLAB `listener` function to monitor event states.

- `Connecting`, `ConnectFailed`, `Connected` – Events related to target computer connection operation by using the Real-Time tab in the Simulink Editor, Simulink Real-Time Explorer, or the `connect` function.
- `Disconnecting`, `Disconnected` – Events related to target computer disconnection operation by using the Real-Time tab in the Simulink Editor, Simulink Real-Time Explorer, or the `disconnect` function.
- `Installing`, `InstallFailed`, `Installed` – Events related to real-time application installation on a target computer by using the `install` function.
- `Loading`, `LoadFailed`, `Loaded` – Events related to real-time application load on a target computer by using the Real-Time tab in the Simulink Editor, Simulink Real-Time Explorer, or the `load` function.
- `Starting`, `StartFailed`, `Started` – Events related to real-time application start on a target computer by using the Real-Time tab in the Simulink Editor, Simulink Real-Time Explorer, or the `start` function.
- `Stopping`, `StopFailed`, `Stopped` – Events related to real-time application stop on a target computer by using the Real-Time tab in the Simulink Editor, Simulink Real-Time Explorer, or the `stop` function.
- `Rebooting`, `RebootFailed`, `RebootIssued` – Events related to target computer reboot operation by using the Simulink Real-Time Explorer or the `reboot` function.
- `UpdateBegin`, `UpdateMessage`, `UpdateFailed`, `UpdateCompleted` – Events related to target computer RTOS software update operation by using the Simulink Real-Time Explorer or the `update` function.

- `SetIPAddressBegin`, `SetIPAddressFailed`, `SetIPAddressCompleted` – Events related to target computer IP address change operation by using the Simulink Real-Time Explorer or the `setipaddr` function.
- `StartupAppChanged` – Event related to target computer startup application change operation by using the Simulink Real-Time Explorer or the `setStartupApp` or `clearStartupApp` functions.
- `StopTimeChanged` – Event related to real-time application stop time change operation by using the Simulink Real-Time Explorer or the `setStopTime` function.

Object Functions

addInstrument	Add instrument object to target object
clearStartupApp	Clear startup application selection on target computer
connect	Connect MATLAB to target computer
deleteProfilerData	Delete execution profiler data from target computer
disconnect	Disconnect MATLAB from target computer
getAvailableProfile	Get information about available execution profiler data
getProfilerData	Retrieve profile data object
getStartupApp	Get information about startup application configuration on target computer
getparam	Read value of observable parameter in real-time application
install	Install real-time application on target computer
load	Deploy to target and load real-time application to target computer
reboot	Restart target computer
removeAllInstruments	Remove instrument objects from target object
removeInstrument	Remove selected instrument object from target object
resetProfiler	Reset profiling service state to Ready
setipaddr	Set IP address and netmask on the target computer
setStartupApp	Configure startup real-time application for target computer
setStopTime	Configure stop time for real-time application
setparam	Change value of tunable parameter in real-time application
start	Start execution of real-time application on target computer
startProfiler	Start profiling service on target computer
status	Get status of real-time application on target computer
stop	Stop execution of real-time application on target computer
stopProfiler	Stop profiling service on target computer
update	Update RTOS version on target computer

Examples

Create Target Object for Default Target Computer

In this example, you create a target object that represents the default target computer.

- Create target object `tg` for default target computer by using configured name for default target computer. You can select the default target computer by using Simulink Real-Time Explorer.

```
tg = slrealtime
```

In this example, you create a target object that represents target computer `TargetPC1`.

- Create target object `tg` for default target computer by using explicit name for default target computer.

```
tg = slrealtime('TargetPC1')
```

Build and Run Real-Time Application

Build and download `slrt_ex_osc` and execute the real-time application.

Open, build, and download the real-time application:

```
model = 'slrt_ex_osc';
open_system(model);
rtwbuild(model);
tg = slrealtime('TargetPC1');
load(tg,model);
start(tg);
```

See Also

“Target Computer Command-Line Interface” | `ProfilerData` | `Target.FileLog` | `Target.ptpd`

Topics

“Parameter Tuning and Data Logging”
“Blocks Whose Outputs Depend on Inherited Sample Time”
“Target and Application Objects”

Introduced in R2020b

addInstrument

Package: slrealtime

Add instrument object to target object

Syntax

```
addInstrument(target_object,instrument_object)
addInstrument(target_object,instrument_object,'updateWhileRunning')
```

Description

addInstrument(target_object,instrument_object) adds an instrument object to the target object. Make sure that you add a signal to the instrument object before you add the instrument to the target object or no signal is streamed.

addInstrument(target_object,instrument_object,'updateWhileRunning') adds an instrument object to the target object and updates the target connection, even if the real-time application is running. Make sure that you add a signal to the instrument object before you add the instrument to the target object or no signal is streamed.

Examples

Add Instrument Object

Create a target object. Build the real-time application. Create the instrument object. Add a signal to the instrument object. Load the real-time application. Add an instrument object to the target object. Start real-time application.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_tank');
hInst = slrealtime.Instrument('slrt_ex_tank');
hInst.addSignal('slrt_ex_tank/Controller',1)
load(tg,'slrt_ex_tank');
addInstrument(tg,hInst);
start(tg);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

instrument_object — Object that represents real-time instrument
object

To create the instrument object, use the Instrument function.

Example: `hInst`

See Also

`Target` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Topics

"Add App Designer App to Inverted Pendulum Model"

Introduced in R2020b

clearStartupApp

Package: slrealtime

Clear startup application selection on target computer

Syntax

```
clearStartupApp(target_object)
```

Description

`clearStartupApp(target_object)` clears the selection of the startup application on the target computer. When this selection is cleared, after booting the RTOS, the target computer waits for commands from the development computer or target computer keyboard (console).

Examples

Clear Startup Application Selection

This example creates a target object, connects MATLAB to the target computer, clears the startup application selection, and reboots the target computer.

```
tg = slrealtime('TargetPC1');
conect(tg);
clearStartupApp(tg);
reboot(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

`Target` | `getStartupApp` | `setStartupApp`

Topics

"Real-Time Application and Target Computer Modes"
"Target Computer Update, Reboot, and Startup Application"

Introduced in R2020b

connect

Package: `slrealtime`

Connect MATLAB to target computer

Syntax

```
connect(target_object)
```

Description

`connect(target_object)` connects MATLAB® to the target computer by using the target object. This connection establishes communication between the development computer and target computer.

Examples

Connect Target Object

Create a target object that represents the target computer. Connect the development computer and target computer by using the target object.

```
tg = slrealtime('TargetPC1');
connect(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

`Target` | `load` | `start` | `stop`

Topics

"Real-Time Application and Target Computer Modes"

Introduced in R2020b

deleteProfilerData

Package: slrealtime

Delete execution profiler data from target computer

Syntax

```
deleteProfilerData(target_object,'-all')
deleteProfilerData(target_object,app_name)
```

Description

deleteProfilerData(target_object,'-all') deletes execution profiler data from all of the installed real-time applications on the target computer.

For information about the availability of log data, see list.

deleteProfilerData(target_object,app_name) deletes all of the execution profiler data from the selected real-time applications on the target computer.

Examples

Delete Profiler Data for All Applications

For target computer object tg with execution profiler data available for real-time applications, delete profiler data for all applications.

```
deleteProfilerData(tg,'-all')
```

Delete Profiler Data for Selected Application

For target computer object tg with execution profiler data available for real-time application my_app, delete profiler data for application my_app.

```
deleteProfilerData(tg,'my_app')
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

app_name — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

See Also

Enable Profiler | `ProfilerData` | `Target` | `getProfilerData` | `resetProfiler` | `startProfiler` | `stopProfiler`

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

disconnect

Package: slrealtime

Disconnect MATLAB from target computer

Syntax

disconnect(target_object)

Description

disconnect(target_object) disconnects MATLAB from the target computer by using the target object.

Examples

Disconnect Target Object

Create a target object that represents the target computer. Connect the development computer and target computer by using the target object. Disconnect the target computer.

```
tg = slrealtime('TargetPC1');
connect(tg);
disconnect(tg);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Target | load | start | stop

Topics
“Real-Time Application and Target Computer Modes”

Introduced in R2020b

getAvailableProfile

Package: `slrealtime`

Get information about available execution profiler data

Syntax

```
prof_info = getAvailableProfile(target_object,app_name)
prof_info = getAvailableProfile(target_object,'-all')
```

Description

`prof_info = getAvailableProfile(target_object,app_name)` gets information about execution profile data that is available for the specified real-time application on the target computer.

`prof_info = getAvailableProfile(target_object,'-all')` gets information about execution profile data that is available for all real-time applications on the target computer.

Examples

Get Available Profiler Data Information for Application

For target computer object `tg`, get information about available execution profiler data for application `my_app`.

```
my_prof_info = getAvailableProfile(tg, 'my_app');
```

Get Available Profiler Information for All Applications

For target computer object `tg`, get information about all available execution profiler data for installed applications.

```
my_prof_info = getAvailableProfile(tg, '-all');
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`app_name` — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

Output Arguments

`prof_info` — Info about application or applications with profiler data available

string scalar | array of strings

If no profiler data is available, the `prof_info` is an empty string. If profiler data is available for the selected real-time application, the returned string contains the application name. If profiler data is available for multiple applications and you use the `'-all'` option, the return value is an area of strings with each string containing an application name..

See Also

Enable Profiler | `ProfilerData` | `Target` | `deleteProfilerData` | `resetProfiler` | `startProfiler` | `stopProfiler`

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

getparam

Package: slrealtime

Read value of observable parameter in real-time application

Syntax

```
value = getparam(target_object, block_path, parameter_name)
value = getparam(target_object, '', parameter_name)
```

Description

`value = getparam(target_object, block_path, parameter_name)` returns the value of block parameter *`parameter_name`* in block *`block_path`* from the real-time application that is loaded on the target computer.

`value = getparam(target_object, '', parameter_name)` returns the value of global parameter *`parameter_name`*.

Examples

Get Block Parameter by Using Parameter and Block Names

This example builds a real-time application from model `slrt_ex_testmodel`, loads the application on the target computer, and gets the value of block parameter `'Amplitude'` of block `'Signal Generator'`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
getparam(tg,'slrt_ex_testmodel/Signal Generator','Amplitude')
```

```
ans =

     4
```

Get Global Parameter by Using Scalar Parameter Name

This example assumes that in model `slrt_ex_testmodel` you previously created a variable `Freq` and assigned the `Frequency` parameter value to `Freq`. The example builds a real-time application from model `slrt_ex_testmodel`, loads the application on the target computer, and gets the value of MATLAB variable `'Freq'`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg, 'slrt_ex_testmodel');
getparam(tg,'','Freq')
```

```
ans =

     20
```

Get Global Parameter by Using Parameter Structure Name

This example creates an array of gain values and assigns the gain parameters to its elements. The example builds a real-time application from model `slrt_ex_testmodel`, loads the application on the target computer, and gets the value of parameter structure `'oscp'`.

```
oscp.G0 = 1000000;
oscp.G1 = 400;
oscp.G2 = 1000000;
set_param('slrt_ex_testmodel/Gain','Gain','oscp.G0');
set_param('slrt_ex_testmodel/Gain1','Gain','oscp.G1');
set_param('slrt_ex_testmodel/Gain2','Gain','oscp.G2');
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
getparam(tg,'','oscp')
```

```
ans =

    G0: 1000000
    G1: 400
    G2: 1000000
```

Input Arguments

`target_object` — Object that represent target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`block_path` — Hierarchical name of the originating block

character vector | string scalar | cell array of character vectors or strings

The *`block_path`* values can be:

- Empty character vector (`''`) or empty string scalar (`""`) for base or model workspace variables
- Character vector or string scalar string for block path to parameters in the top model
- Cell array of character vectors or string scalars for model block arguments

Example: `''`, `'Gain1'`, `{'top/model','sub/model'}`

`parameter_name` — Name of the parameter

character vector | string scalar

The parameter can designate either a block parameter or a global parameter that provides the value for a block parameter. The block parameter or MATLAB variable must be observable to be accessible through the parameter name.

Note Simulink Real-Time does not support parameters of multiword data types.

Example: `'Gain'`, `'oscp.G1'`, `'oscp'`, `'G2'`

Output Arguments

`value` — Value of parameter

scalar | complex | structure

Simulink Real-Time does not support parameters of multiword data types.

See Also

`Target` | `load` | `setparam` | `start` | `stop`

Topics

"Tunable Block Parameters and Tunable Global Parameters"
"Troubleshoot Parameters Not Accessible by Name"

Introduced in R2020b

getProfilerData

Package: `slrealtime`

Retrieve profile data object

Syntax

```
profiler_object = getProfilerData(target_object)
profiler_object = getProfilerData(target_object);
```

Description

`profiler_object = getProfilerData(target_object)` downloads the profiler files from the target computer to the development computer and assigns the data to the `profiler_object`. It displays an execution profile plot and a code execution profiling report.

The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The Code Execution Profiling Report lists the model sections. The numbers underneath the bars indicate the processor cores.

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

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

`profiler_object = getProfilerData(target_object);` downloads the profiler files from the target computer to the development computer and assigns the data to `profiler_object`. To display the profiler results, call the `plot` and `report` functions with the `profiler_object` as the argument.

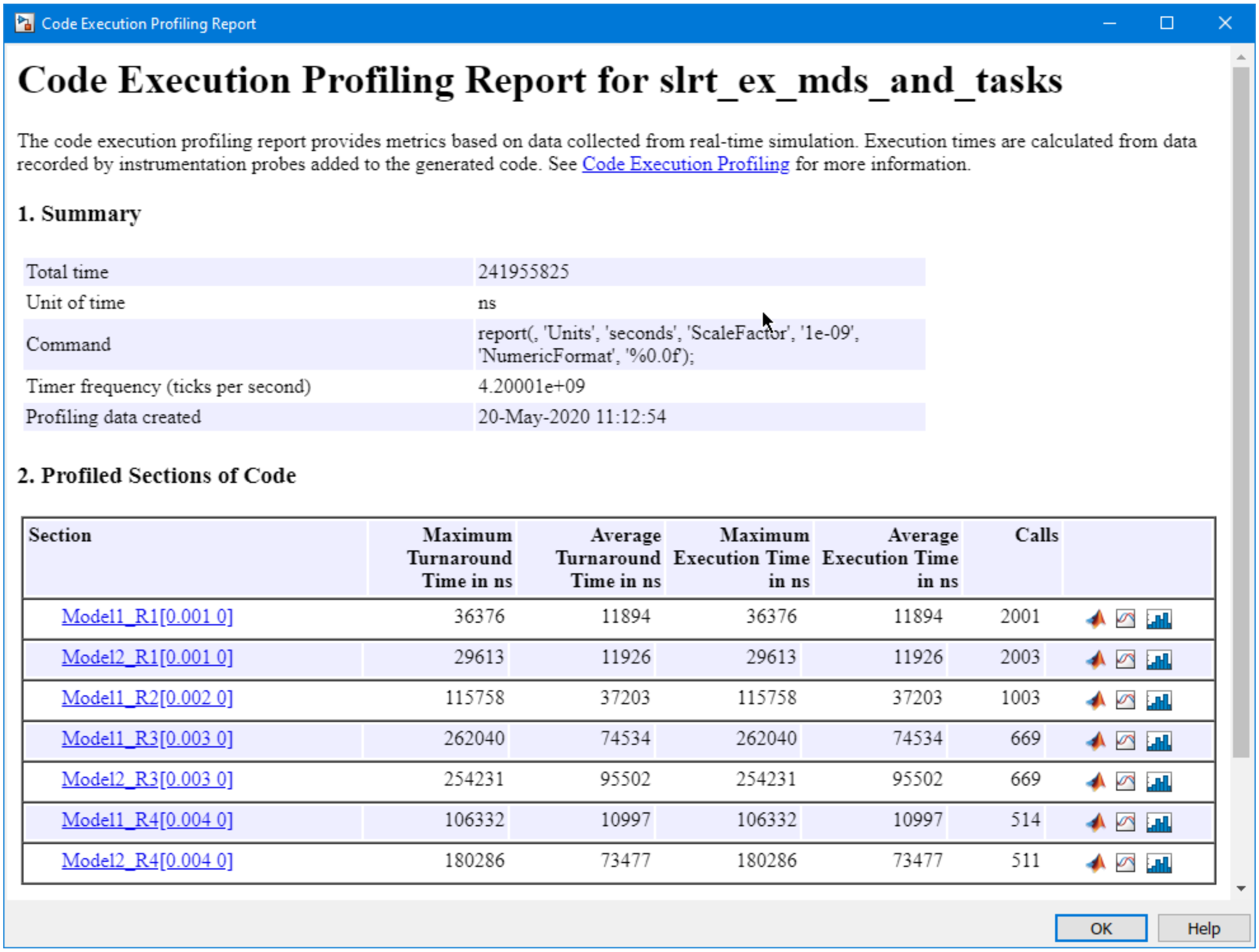
Examples

Run Profiler and Implicitly Display Profiler Data

This example starts the profiler, stops the profiler, and displays execution profile data. The real-time application `slrt_ex_mds_and_tasks` is already loaded.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);
start(tg);
```

```
stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg)

Processing data on target computer, please wait ...
Transferring data from target computer to host computer, please wait ...
Processing data on host computer, please wait ...

Code execution profiling data for model slrt_ex_mds_and_tasks.
```



Code Execution Profiling Report for slrt_ex_mds_and_tasks

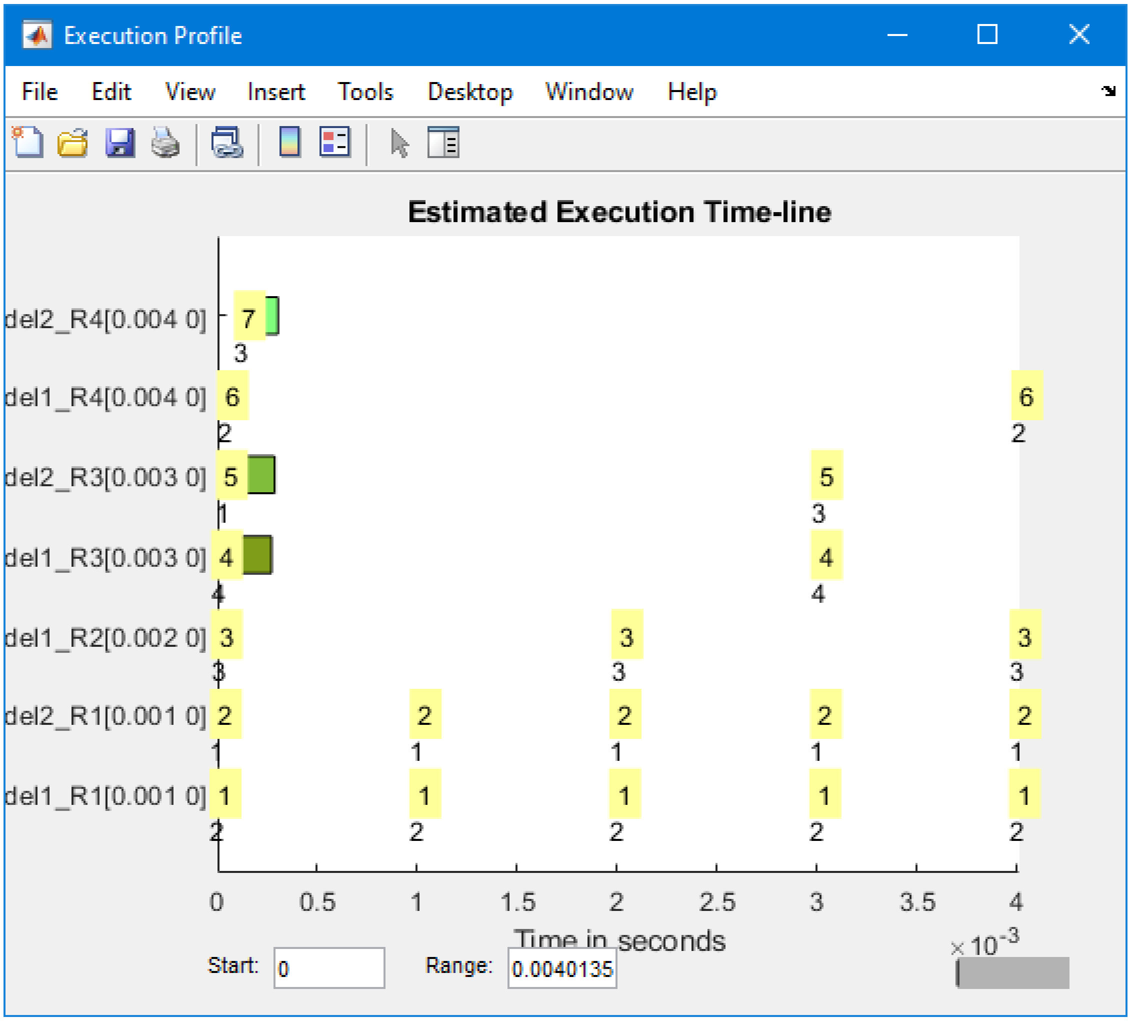
The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

Total time	241955825
Unit of time	ns
Command	report(, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	4.20001e+09
Profiling data created	20-May-2020 11:12:54

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Model1_R1[0.001 0]	36376	11894	36376	11894	2001
Model2_R1[0.001 0]	29613	11926	29613	11926	2003
Model1_R2[0.002 0]	115758	37203	115758	37203	1003
Model1_R3[0.003 0]	262040	74534	262040	74534	669
Model2_R3[0.003 0]	254231	95502	254231	95502	669
Model1_R4[0.004 0]	106332	10997	106332	10997	514
Model2_R4[0.004 0]	180286	73477	180286	73477	511



Run Profiler and Explicitly Display Profiler Data

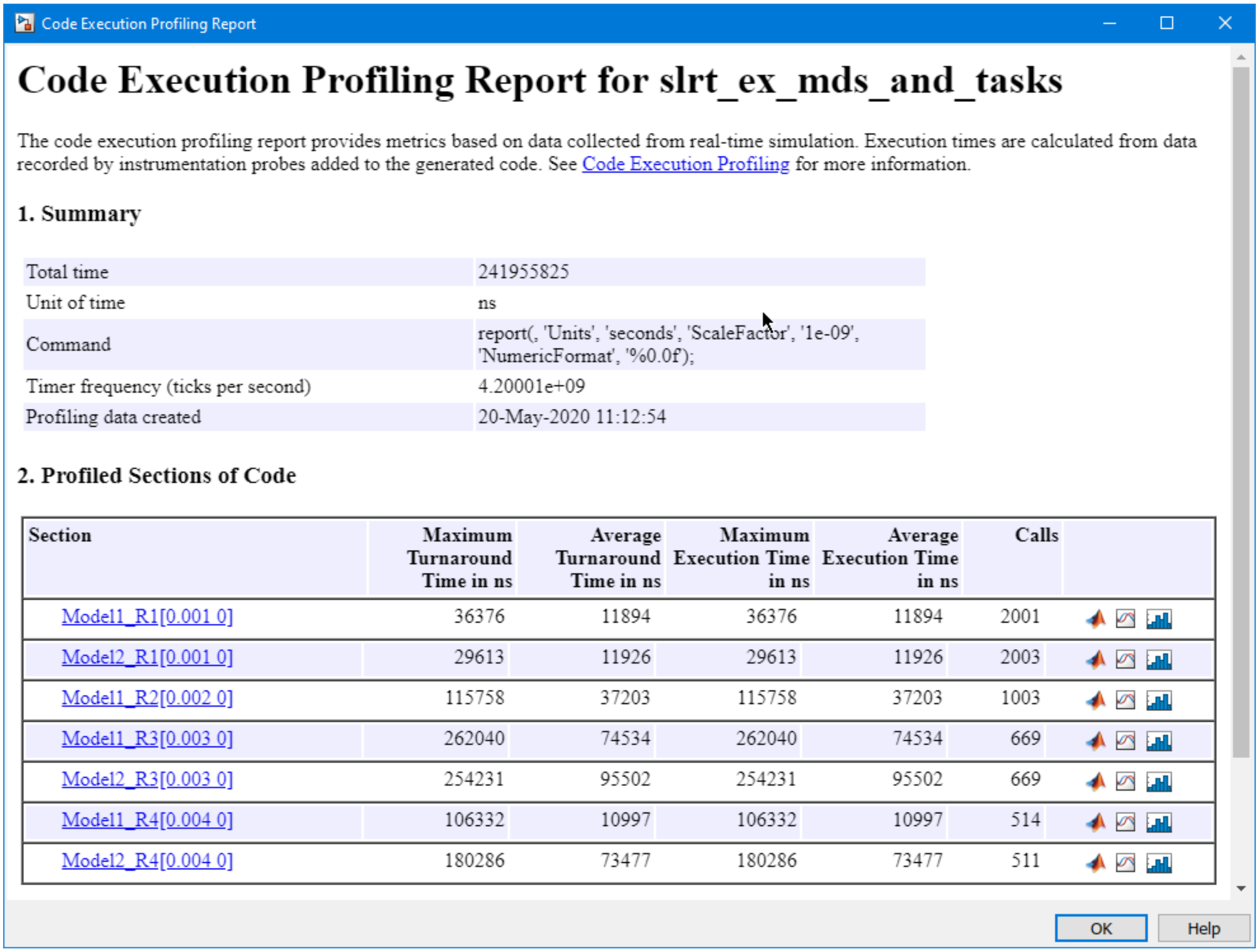
Starts the profiler, stops the profiler, and retrieves results data. Calls `report` and `plot` on the results data. The real-time application `slrt_ex_mds_and_tasks` is already loaded.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);
```

```
rocessing data on target computer, please wait ...
Transferring data from target computer to host computer, please wait ...
Processing data on host computer, please wait ...
```

```
Code execution profiling data for model slrt_ex_mds_and_tasks.
```

```
report(profiler_object);
```



Code Execution Profiling Report for slrt_ex_mds_and_tasks

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

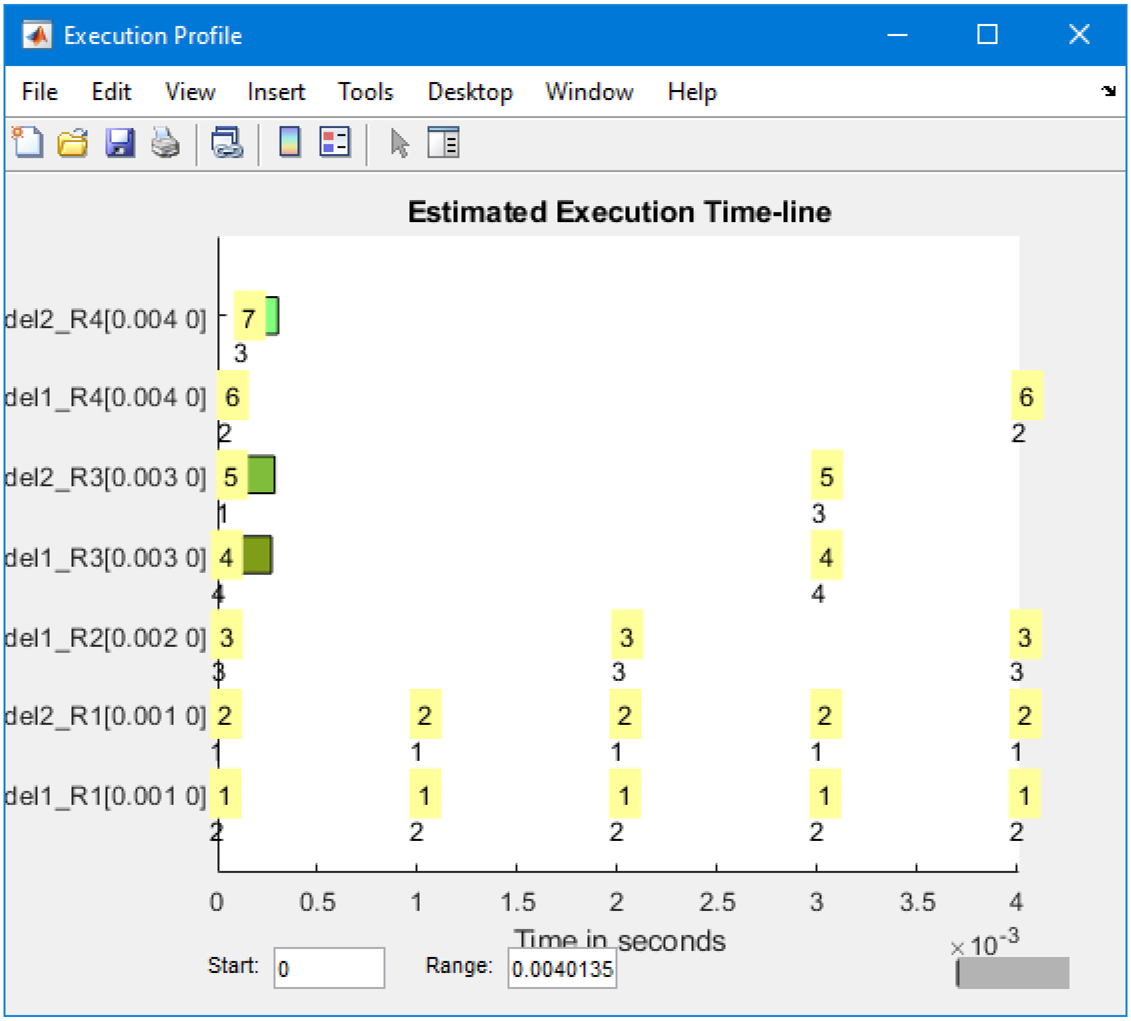
1. Summary

Total time	241955825
Unit of time	ns
Command	report(, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	4.20001e+09
Profiling data created	20-May-2020 11:12:54

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Model1_R1[0.001 0]	36376	11894	36376	11894	2001
Model2_R1[0.001 0]	29613	11926	29613	11926	2003
Model1_R2[0.002 0]	115758	37203	115758	37203	1003
Model1_R3[0.003 0]	262040	74534	262040	74534	669
Model2_R3[0.003 0]	254231	95502	254231	95502	669
Model1_R4[0.004 0]	106332	10997	106332	10997	514
Model2_R4[0.004 0]	180286	73477	180286	73477	511

```
plot(profiler_object);
```



Start: 0

Range: 0.0040135

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`profiler_object` — Object that contains profiler result
structure

MATLAB variable that you can use to access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has these fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters** dialog box. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation** > **Verification** > **Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

Enable Profiler | `ProfilerData` | `Target` | `resetProfiler` | `stopProfiler`

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

getStartupApp

Package: slrealtime

Get information about startup application configuration on target computer

Syntax

```
getStartupApp(target_object)
```

Description

getStartupApp(target_object) gets information about the startup application configuration on the target computer. If you select a startup application, after booting the RTOS, the target computer loads and starts the startup application.

Examples

Get Startup Application for Target Object

For target object tg, get information about the startup real-time application configuration. The getStartupApplication function returns the name of the application as a character vector.

```
tg = slrealtime('TargetPC1');
conect(tg);
load(tg,'slrt_ex_ExecutionProfAndConc')
setStartupApp(tg,'slrt_ex_ExecutionProfAndConc')
getStartupApp(tg)
```

```
ans =

    'slrt_ex_ExecutionProfAndConc'
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Target | clearStartupApp | setStartupApp

Topics
“Real-Time Application and Target Computer Modes”
“Target Computer Update, Reboot, and Startup Application”

Introduced in R2020b

install

Package: `slrealtime`

Install real-time application on target computer

Syntax

```
install(target_object,app_name)
install(target_object,app_name,'force')
```

Description

`install(target_object,app_name)` installs a real-time application on the target computer if the application does not exist on the target computer or if the checksum of the previously installed application does not match the application in the `install` command.

`install(target_object,app_name,'force')` installs a real-time application on the target computer without checking for a previously installed application.

Examples

Install Application on Target Computer

Install the real-time application `slrt_ex_osc` on the target computer `TargetPC1`, represented by target object `tg`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_osc');
install(tg,'slrt_ex_osc');
```

Force Install of Application on Target Computer

Force an installation of the real-time application `slrt_ex_osc` into target computer `TargetPC1`, represented by target object `tg`. By using the `force` option, the function installs the real-time application on the target computer without checking for a previously installed application or checking whether a previously installed version of the application is up to date.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_osc');
install(tg,'slrt_ex_osc','force');
```

Input Arguments

`target_object` — Object that represent target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`app_name` — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

See Also

`Target` | `start` | `stop`

Topics
"Real-Time Application and Target Computer Modes"

Introduced in R2020b

load

Package: slrealtime

Deploy to target and load real-time application to target computer

Syntax

```
load(target_object,app_name)
```

Description

load(target_object,app_name) deploys and loads the application *app_name* onto the target computer represented by the *target_object*.

The load command checks whether Simulink Real-Time software is connected to the RTOS on the target computer. If not connected, the load connects to the target computer before loading the real-time application.

You also can load the real-time application from the RTOS command line. For more information, see “Execute Target Computer RTOS Commands at Target Computer Command Line” and “Target Computer Command-Line Interface”.

If you are running the real-time application in standalone mode, instead of load, consider using the install function and the setStartupApp function. For more information about Simulink Real-Time modes, see “Real-Time Application and Target Computer Modes”.

Examples

Load Application

Load the real-time application slrt_ex_osc on the target computer TargetPC1, represented by target object tg. Start the application.

Get the target object, and then build the real-time application.

```
tg = slrealtime('TargetPC1');
```

Build the real-time application.

```
rtwbuild('slrt_ex_osc');
```

Load the real-time application.

```
load(tg,'slrt_ex_osc');
```

Start the application.

```
start(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`app_name` — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

See Also

`Target` | `start` | `stop`

Topics
"Real-Time Application and Target Computer Modes"
"Execute Target Computer RTOS Commands at Target Computer Command Line"
"Target Computer Command-Line Interface"

Introduced in R2020b

reboot

Package: `slrealtime`

Restart target computer

Syntax

```
reboot(target_object)
```

Description

`reboot(target_object)` restarts the target computer that is represented by the *`target_object`*. When you start the target computer, it boots the RTOS. The target computer boots in standalone mode. For more information, see “Real-Time Application and Target Computer Modes”.

You also can reboot the target computer from the RTOS command line. For more information, see “Execute Target Computer RTOS Commands at Target Computer Command Line” and “Target Computer Command-Line Interface”.

Examples

Restart Target Computer `'TargetPC1'`

Get a target object and restart the target computer that it represents.

Get target object for target computer `'TargetPC1'` and connect Simulink Real-Time to the target computer.

```
tg = slrealtime('TargetPC1');
```

Restart target computer.

```
reboot(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

`Target`

Topics

“Real-Time Application and Target Computer Modes”
“Execute Target Computer RTOS Commands at Target Computer Command Line”

“Target Computer Command-Line Interface”

Introduced in R2020b

removeAllInstruments

Package: `slrealtime`

Remove instrument objects from target object

Syntax

`removeAllInstruments(target_object)`

Description

`removeAllInstruments(target_object)` removes the connections to instrument objects from the target object.

Examples

Remove Instrument Objects

Create a target object. Build the real-time application. Create the instrument object. Add a signal to the instrument object. Load the real-time application. Add an instrument object to the target object. Start real-time application. Remove instrument objects from target object.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_tank');
hInst = slrealtime.Instrument('slrt_ex_tank');
hInst.addSignal('slrt_ex_tank/Controller',1)
load(tg,'slrt_ex_tank');
addInstrument(tg,hInst);
start(tg);
removeAllInstruments(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

`Target` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Topics
"Add App Designer App to Inverted Pendulum Model"

Introduced in R2020b

removeInstrument

Package: `slrealtime`

Remove selected instrument object from target object

Syntax

```
removeInstrument(target_object,instrument_object)
```

Description

`removeInstrument(target_object,instrument_object)` removes the connection to the selected instrument object from the target object.

Examples

Remove Selected Instrument Object

Create a target object. Build the real-time application. Create the instrument object. Add a signal to the instrument object. Load the real-time application. Add an instrument object to the target object. Start real-time application. Remove the selected instrument object from target object.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_tank');
hInst = slrealtime.Instrument('slrt_ex_tank');
hInst.addSignal('slrt_ex_tank/Controller',1)
load(tg,'slrt_ex_tank');
addInstrument(tg,hInst);
start(tg);
removeInstrument(tg,hInst);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

See Also

`Target` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Topics

"Add App Designer App to Inverted Pendulum Model"

Introduced in R2020b

resetProfiler

Package: `slrealtime`

Reset profiling service state to Ready

Syntax

```
resetProfiler(target_object)
```

Description

`resetProfiler(target_object)` resets the profiling service state to Ready and deletes any data that the profiler has collected.

When you start a real-time application, the profiler resets itself.

Examples

Reset Profiler

Start profiling execution, and then reset the profiler. The real-time application is already running.

```
tg = slrealtime('TargetPC1');
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
    'examples', 'slrt_ex_mds_and_tasks'))
rtwbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);

% start profiler before starting application

start(tg);

resetProfiler(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

Enable Profiler | `ProfilerData` | `Target`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2020b

setipaddr

Package: slrealtime

Set IP address and netmask on the target computer

Syntax

setipaddr(target_object,'ipaddr','netmask')

Description

setipaddr(target_object,'ipaddr','netmask') sets the IP address and netmask on the target computer. If the *netmask* argument is omitted, the default value is '255.255.255.0'.

Examples

Set IP Address on Target Computer

For target object tg, set the target computer IP address to '10.10.10.10' and the netmask to '255.255.255.0'. These values are retained by the target computer.

```
tg = slrealtime('TargetPC1');
setipaddr(tg,'10.10.10.10','255.255.255.0');
reboot(tg);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

ipaddr — IP address of target computer
character vector | string scalar

This value sets the IP address of the target computer.

Example: '10.10.10.10'

netmask — Netmask of target computer
'255.255.255.0' (default) | character vector | string scalar

This value sets the netmask of the target computer.

Example: '255.255.255.0'

See Also

Target | load | start | stop

Topics
“Real-Time Application and Target Computer Modes”

Introduced in R2020b

setparam

Package: slrealtime

Change value of tunable parameter in real-time application

Syntax

```
setparam(target_object, block_path, parameter_name, parameter_value)
setparam(target_object, '', parameter_name, parameter_value)
```

Description

`setparam(target_object, block_path, parameter_name, parameter_value)` sets the value of a tunable block parameter to a new value. Specify the block parameter by the block name and the parameter name.

`setparam(target_object, '', parameter_name, parameter_value)` sets the value of the tunable global parameter to a new value. Specify the global parameter by the MATLAB variable name.

Examples

Set Block Parameter by Parameter and Block Names

Set the value of the block parameter `'Amplitude'` of the block `'Signal Generator'` to `5`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
setparam(tg,'slrt_ex_testmodel/Signal Generator','Amplitude',5)
```

Sweep Block Parameter Values

Sweep the value of the block parameter `'Amplitude'` of the block `'Signal Generator'` by steps of `2`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
for i = 1 : 3
    setparam(tg,'slrt_ex_testmodel/Signal Generator','Amplitude',(i*2))
end
```

Set Global Parameter by Scalar Parameter Name

Set the value of the MATLAB variable `'Freq'` to `30`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
setparam(tg,'','Freq',30)
```

Set Global Parameter by Parameter Structure Field Name

Set the value of the MATLAB variable `'oscp.G2'` to `10000000`.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_testmodel');
load(tg,'slrt_ex_testmodel');
setparam(tg,'','oscp.G2',10000000)
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`block_path` — Hierarchical name of the originating block
character vector | string scalar | cell array of character vectors or strings

The *`block_path`* values can be:

- Empty character vector (`''`) or empty string scalar (`""`) for base or model workspace variables
- Character vector or string scalar string for block path to parameters in the top model
- Cell array of character vectors or string scalars for model block arguments

Example: `''`, `'Gain1'`, `{'top/model','sub/model'}`

`parameter_name` — Name of the parameter
character vector | string scalar

The parameter can designate either a block parameter or a global parameter that provides the value for a block parameter. The block parameter or MATLAB variable must be observable to be accessible through the parameter name.

Note Simulink Real-Time does not support parameters of multiword data types.

Example: `'Gain'`, `'oscp.G1'`, `'oscp'`, `'G2'`

`parameter_value` — New parameter value
number | character vector | string scalar | complex | structure

New value with data type as required by parameter.

Example: 1

See Also

`Target` | `getparam` | `load` | `start` | `stop`

Topics

“Tunable Block Parameters and Tunable Global Parameters”
“Troubleshoot Parameters Not Accessible by Name”

Introduced in R2020b

setStartupApp

Package: slrealtime

Configure startup real-time application for target computer

Syntax

```
setStartupApp(target_object,app_name)
```

Description

`setStartupApp(target_object,app_name)` configures the target computer to run the selected real-time application on startup.

Examples

Configure Startup Application

Create target object, connect to target computer, and configure the startup application for the target computer. When you reboot or restart the target computer, after the target computer boots the RTOS, the startup application is loaded and runs.

```
tg = slrealtime('TargetPC1');
connect(tg);
setStartupApp(tg,'slrt_ex_osc');
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

app_name — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: 'slrt_ex_osc'

See Also

Target | clearStartupApp | getStartupApp

Topics
"Real-Time Application and Target Computer Modes"

Introduced in R2020b

setStopTime

Package: slrealtime

Configure stop time for real-time application

Syntax

```
setStopTime(target_object,stopTime)
```

Description

setStopTime(target_object,stopTime) configures the stop time value for the real-time application that is loaded on the target computer. This value replaces the stop time value from the model that built the application.

Examples

Configure Stop Time

Create the target object. Load the real-time application on the target computer. Configure the stop time for the real-time application.

```
tg = slrealtime('TargetPC1');
load(tg,'slrt_ex_osc')
setStopTime(tg,10);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

stopTime — Application stop time in seconds
double

Selects the stop time value in seconds for the real-time application. This value is a real-time application option and is retained on the target computer.

Example: 10

See Also

Target | start | stop

Topics
"Real-Time Application and Target Computer Modes"

Introduced in R2020b

slrealtime

Package: slrealtime

Interface for managing target computer

Syntax

```
target_object = slrealtime
target_object = slrealtime(target_name)
```

Description

`target_object = slrealtime` constructs a target object representing the default target computer. Select the default target computer by using the `slrtExplorer`.

`target_object = slrealtime(target_name)` constructs a target object representing the target computer designated by `target_name`.

Examples

Default Target Computer

Create a target object that communicates with the default target computer. Select the default target computer by using the `slrtExplorer`.

```
target_object = slrealtime('TargetPC1');
```

Specific Target Computer

Create a target object that communicates with target computer `TargetPC1`. Report the status of the target computer. In this case, the target computer is not connected to the development computer.

```
target_object = slrealtime('TargetPC1')
```

```
Target: TargetPC1
   Connected            = No
```

Input Arguments

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

Example: `'TargetPC1'`

Data Types: `char` | `string`

Output Arguments

`target_object` — Object that represent target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

`Target` | `Targets`

Introduced in R2020b

start

Package: slrealtime

Start execution of real-time application on target computer

Syntax

```
start(target_object,Name-Value Pair Arguments)
```

Description

`start(target_object,Name-Value Pair Arguments)` starts execution of the real-time application that is loaded on the target computer, which is represented by the *`target_object`*. Before using this method, you must create and load the real-time application on the target computer. If a real-time application is running, issuing a `start` command generates an error.

You can also start the real-time application from the RTOS command line. For more information, see “Execute Target Computer RTOS Commands at Target Computer Command Line” and “Target Computer Command-Line Interface”.

Examples

Start Execution of Real-Time Application

Start execution of the real-time application that is loaded on the target computer, which is represented by the target object `tg`.

```
tg = slrealtime('TargetPC1');
load(tg, 'my_xpctank');
start(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `start(tg,'LogLevel',info)`

`LogLevel` — System log message level
`info` (default) | `trace` | `debug` | `warning` | `error` | `fatal`

Selects filtering level that limits Simulink Real-Time target computer system messages that appear in the system log. For more information, see “Simulink Real-Time Options Pane”.

Example: info

`PollingThreshold` — Threshold value for polling

`100` (default) | int32

The real-time application is clocked by a timer interrupt, unless the base sample rate is equal to or below the polling threshold (default is 100 µs). If the base sample rate is less than or equal to the threshold, the real-time application is clocked in polling mode.

Example: `100`

`FileLogMaxRuns` — Number of file logs retained

`1` (default) | int

Select the number of file logs to retain when logs are stored on the target computer instead of uploaded to the development computer after each simulation run.

Example: `1`

`StopTime` — Real-time application stop time

`StopTime` config set value (default)

Select stop time value for the real-time application.

Example: `Inf`

`ReloadOnStop` — Reload real-time application

`false` (default) | `true`

Direct Simulink Real-Time to reload the real-time application on the target computer after the application stops.

Example: `false`

`AutoImportFileLog` — Configure file log import

`true` (default) | `false`

Select whether the file log data is uploaded the Simulation Data Inspector on the development computer after the real-time application stops.

Example: `true`

`ExportToBaseWorkspace` — Configure file log export

`true` (default) | `false`

Select whether the file log data is uploaded the Simulink base workspace on the development computer after the real-time application stops

Example: `true`

See Also

`Target` | `load` | `stop`

Topics

“Real-Time Application and Target Computer Modes”

“Execute Target Computer RTOS Commands at Target Computer Command Line”
“Target Computer Command-Line Interface”

Introduced in R2020b

startProfiler

Package: slrealtime

Start profiling service on target computer

Syntax

startProfiler(target_object,app_name)

Description

startProfiler(target_object,app_name) starts the profiler on the target computer. You can start the profiler before or after you load the real-time application on the target computer. Before you start the application, you must start the profiler.

The startProfiler function affects the value of the *target_object* property ProfilerStatus.

- Ready status indicates that the *target_object* exists, no profiling data is available, and the startProfiler function has not been called.
- StartRequested status indicates that the *target_object* exists, no profiling data is available, the startProfiler function has started the profiler, and the real-time application is not loaded.
- Running status indicates that the *target_object* exists, profiling data is being collected, the startProfiler function has started the profiler, and the real-time application is loaded and running.
- DataAvailable status indicates that the *target_object* exists, profiling data is available, and the real-time application and the profiler have stopped.

Examples

Profile Execution of Real-Time Application

Build the real-time application slrt_ex_ExecutionProfAndConc. Load the real-time application. Start the profiler. Start the application.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_ExecutionProfAndConc');
load(tg,'slrt_ex_ExecutionProfAndConc');
startProfiler(tg);

% start profiler before starting application

start(tg);
```

Check Profiler Status from Target Object Property

Build the real-time application slrt_ex_ExecutionProfAndConc. Load the application. Check the profiler status from the target object property ProfilerStatus.

```
tg = slrealtime('TargetPC1');
rtwbuild(''slrt_ex_ExecutionProfAndConc');
load(tg,''slrt_ex_ExecutionProfAndConc');
tg.ProfilerStatus
```

```
ans =

    'Ready'
```

Start the profiler, and then start the application.

```
startProfiler(tg);
```

```
% start profiler before starting application
```

```
start(tg);
```

After the application stops, check the profiler status.

```
tg.ProfilerStatus
```

```
ans =

    'DataAvailable'
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

app_name — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

See Also

Enable Profiler | `ProfilerData` | `Target` | `getProfilerData` | `resetProfiler` | `stopProfiler`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2020b

status

Package: slrealtime

Get status of real-time application on target computer

Syntax

status(target_object)

Description

status(target_object) returns the status of the real-time application on the target computer. The status values are:

- loading — The real-time application is loading on the target computer.
- loaded — The real-time application is loaded on the target computer.
- running — The real-time application is running on the target computer.
- terminating — The real-time application is terminating on the target computer.
- stopped — The real-time application has stopped on the target computer.
- modelError — An error has occurred in the real-time application on the target computer.

Examples

Get Application Status

Get the status of the real-time application that is loaded on the target computer, which is represented by the target object tg.

```
tg = slrealtime('TargetPC1');
load(tg, 'my_xpctank');
status(tg);
```

```
ans =

    'loaded'
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Target | load | start | stop

Topics
“Real-Time Application and Target Computer Modes”

Introduced in R2020b

stop

Package: slrealtime

Stop execution of real-time application on target computer

Syntax

```
stop(target_object)
```

Description

stop(target_object) stops execution of the real-time application that is running on the target computer, which is represented by the *target_object*. Before using this method, you must create, load, and start the real-time application on the target computer. If a real-time application is loaded on the target computer, but is not running, this command unloads the application.

You can also stop the real-time application from the RTOS command line. For more information, see “Execute Target Computer RTOS Commands at Target Computer Command Line” and “Target Computer Command-Line Interface”.

Examples

Stop Execution of Real-Time Application

Stop execution of the real-time application that is running on the target computer, which is represented by the target object tg.

```
tg = slrealtime('TargetPC1');
load(tg, 'my_xpctank');

% If stop occurs when application is loaded but not started,
% the application is unloaded (process stops).

start(tg);
stop(tg);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Target | load | start

Topics
“Real-Time Application and Target Computer Modes”
“Execute Target Computer RTOS Commands at Target Computer Command Line”
“Target Computer Command-Line Interface”

Introduced in R2020b

stopProfiler

Package: slrealtime

Stop profiling service on target computer

Syntax

```
stopProfiler(target_object)
```

Description

`stopProfiler(target_object)` stops the profiler from running on the target computer.

If the profiler collected data, the data is available for download to the development computer.

If the profiler did not collect data, the profiler is ready to restart.

If you stop execution of the real-time application, the profiler stops.

Examples

Start and Stop Profiler

Start the profiler, and then start the real-time application. After collecting execution profile data, stop the profiler.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_ExecutionProfAndConc');
load(tg,'slrt_ex_ExecutionProfAndConc');
startProfiler(tg);

% start profiler before starting application

start(tg);

% let application run until its stop time
% or stop the profiler by calling stopProfiler

stopProfiler(tg);
```

At this point, call either the `getProfilerData` function or the `resetProfiler` function.

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

Enable Profiler | `ProfilerData` | `Target` | `getProfilerData` | `resetProfiler` | `startProfiler`

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

update

Package: slrealtime

Update RTOS version on target computer

Syntax

```
update(target_object)
update(target_object,'force',true)
```

Description

update(target_object) updates any out-of-date, not-current version RTOS files on the target computer.

update(target_object,'force',true) forces an update of all RTOS files on the target computer to the current version.

Examples

Update RTOS Version

Create a target object that represents the target computer. Update the RTOS version on the target computer. Connect the development computer and target computer.

```
tg = slrealtime('TargetPC1');
update(tg);
connect(tg);
```

Force Update of RTOS Version

Create a target object that represents the target computer. Force the update of the RTOS version on the target computer. The force option is needed for some RTOS states. Connect the development computer and target computer.

```
tg = slrealtime('TargetPC1');
update(tg,'force',true);
connect(tg);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

`Target` | `load` | `start` | `stop`

Topics

"Real-Time Application and Target Computer Modes"

Introduced in R2020b

Target.FileLog

Target Computer file logger

Description

A `Target.FileLog` object represents the file logger that runs on a target computer and provides access to methods and properties related to the file logger.

The object provides access to methods and properties that:

- Enable and disable the file logger.
- Import file log data and abort import processing.
- Check for available file log data.
- Discard unwanted file log data.

Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name if the characters you type are unique for the property.

Creation

A `Target.FileLog` object is created when you create a `Target` object by using the `slrealtime` command. After you create and connect to the `Target` object, you can access the `Target.FileLog` object. This example creates and connects to `Target` object `tg`, and then starts the file logger on the target computer.

```
tg = slrealtime('TargetPC1');
connect(tg);
enable(tg.FileLog);
```

Properties

`Importing` — File log import status

`0` (not importing) (default) | `1` (importing)

The `Importing` property indicates whether the file logger is importing a file log. When `FileLogger` is enabled, the file logger imports file log data at the end of simulation runs. You can disable the import by setting the `Disable automatic import of file logs` option for the real-time application. For more information, see the `start` function.

Example: `0`

`ImportProgress` — File log import progress percentage

`100` (default) | `0..100` (percent complete)

The `ImportProgress` property indicates the percent completion of file log import.

Example: `100`

LoggingService — File logging service status

STARTING (default) | RUNNING | STOPPING | STOPPED | ERROR

The LoggingService property indicates the file logging service status.

Example: 100

DataAvailable — File log data available status

0 (no data available) (default) | 1 (data available)

The DataAvailable property indicates whether file log data is available for import.

Example: 0

Object Functions

abort	Abort file log data import from target computer
disable	Stop file logging of signal data
discard	Delete file log data from target computer
enable	Start file logging of signal data
list	Get information about available file logs of signal data
import	Import file log data from target computer

Examples

Disable File Log

The disable function disables file logging.

Create a Target object and connect to the target computer. Creating a Target object creates a child Target.FileLog object. Connecting to the target computer provides access to the Target.FileLog object. Disable file logging.

```
tg = slrealtime('TargetPC1');
connect(tg);
disable(tg.FileLog);
```

See Also

Target | abort | disable | discard | enable | import | list

Topics

“Parameter Tuning and Data Logging”
“Signal Logging Basics”

Introduced in R2020b

abort

Package: `slrealtime`

Abort file log data import from target computer

Syntax

```
abort(target_object.FileLog)
```

Description

`abort(target_object.FileLog)` aborts the file log import process.

If a Simulink Real-Time model has File Log blocks, when you load the real-time application, file logging is enabled. This default operation is the same as enabling file logging by using the command `enable`.

To control file logging by using the Enable File Log block, when you load the real-time application load, disable file logging by using the command `disable`.

When your development computer is connected to the target computer and the real-time application stops, the file log data is uploaded to the Simulation Data Inspector. For a standalone target computer that does file logging when not connected, after connecting the development and target computers, upload the file logging data for all available runs from an application by using the command `import(Target.FileLog,'app_name')`.

Examples

Abort File Log Data Import

When you stop a real-time application that is file logging, the file log data is uploaded to the Simulation Data Inspector. You can stop the log upload, but the log data is lost. For target computer object `tg` that is uploading file log data from a real-time application, to stop file log import, type:

```
abort(tg.FileLog)
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

Enable File Log | File Log | `Target` | `disable` | `discard` | `enable` | `import` | `list`

Topics
“Signal Logging Basics”

Introduced in R2020b

disable

Package: slrealtime

Stop file logging of signal data

Syntax

```
disable(target_object.FileLog)
```

Description

disable(target_object.FileLog) stops the operation of File Log blocks that are logging signal data.

If a Simulink Real-Time model has File Log blocks, when the real-time application is loaded, file logging is enabled. This default operation is the same as enabling file logging by using the command enable.

To control file logging by using the Enable File Log block, on real-time application load, disable file logging by using the command disable.

When the development computer is connected to the target computer and the model stops, the file log data is uploaded to the Simulation Data Inspector. For a standalone target computer that does file logging when not connected, after connecting the development and target computers, upload the file logging data for the most recent run by using the command import(Target.FileLog,'app_name').

Examples

Disable File Logging

When you start a real-time application that has one or more File Log blocks, file logging starts. You can stop and restart file logging. For target computer object tg with a real-time application loaded and started, to stop file logging, type:

```
disable(tg.FileLog);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Enable File Log | File Log | Target | abort | discard | enable | import | list

Topics
“Signal Logging Basics”

Introduced in R2020b

enable

Package: slrealtime

Start file logging of signal data

Syntax

```
enable(target_object.FileLog)
```

Description

enable(target_object.FileLog) starts operation of stopped File Log blocks.

If a Simulink Real-Time model has File Log blocks, when the real-time application is loaded, file logging is enabled. This default operation is the same as enabling file logging by using the command enable.

To control file logging with the Enable File Log block, when the real-time application is loaded, disable file logging by using the command disable.

When the development computer is connected to the target computer and the model stops, the file log data is uploaded to the Simulation Data Inspector. For a standalone target computer that does file logging when not connected, after connecting the development and target computers, upload the file logging data for all available runs from an application by using the command import(Target.FileLog,'app_name').

Examples

Enable File Logging

When you start a real-time application that has one or more File Log blocks, file logging starts. You can stop and restart file logging. For target computer object tg with a real-time application loaded and started, to start file logging, type:

```
enable(tg.FileLog);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

Enable File Log | File Log | Target | abort | disable | discard | import | list

Topics
"Signal Logging Basics"

Introduced in R2020b

discard

Package: slrealtime

Delete file log data from target computer

Syntax

```
discard(target_object.FileLog,run_info)
discard(target_object.FileLog,app_name)
discard(target_object.FileLog,run_ids)
```

Description

`discard(target_object.FileLog,run_info)` deletes file log data for the installed real-time applications on the target computer.

For information about availability of log data, see `list`.

`discard(target_object.FileLog,app_name)` deletes all of the file log data for the selected real-time applications on the target computer.

`discard(target_object.FileLog,run_ids)` deletes the file log data for the simulation runs that you select from the real-time applications on the target computer.

Examples

Discard File Log Data for Applications

For target computer object `tg` with simulation run data available for real-time applications, delete file log data for applications.

Get table of available simulation run information. Delete file log data from applications in the available file logs table.

```
my_run_info = list(tg.FileLog);
discard(tg.FileLog,my_run_info);
```

Alternatively, you can get the available file log information and delete the file log data in one step.

```
discard(tg.FileLog,tg.FileLog.list);
```

Discard File Log Data for Selected Application

For target computer object `tg` with simulation run data available for real-time application `my_app`, delete file log data for application `my_app`.

```
discard(tg.FileLog,'my_app');
```

Discard File Log Data for Selected Runs

For target computer object `tg` with simulation run data available for real-time applications `slrt_ex_osc_rt_t` and `slrt_ex_osc`, delete file log data for runs 1 and 2.

Get table of available simulation run information.

```
my_run_info = list(tg.FileLog)
```

```
my_run_info = 

  3×3 table

             Application             StartDate          Size
          __________________    ____________________    _____

    1.    "slrt_ex_osc_rt_t"     12-Dec-2019 21:59:31    94944
    2.    "slrt_ex_osc_rt_t"     12-Dec-2019 21:59:45    84736
    3.    "slrt_ex_osc"          12-Dec-2019 21:59:57    82176
```

Delete file log data from application runs 1 and 2 in the available file logs table.

```
discard(tg.FileLog,1:2);
```

Input Arguments

`target_object` — Represent target computer
object

Provides access to methods that manipulate the target computer properties.

app_name — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

`run_info` — Structure of information about file log runs
struct

The *`run_info`* structure is a MATLAB table that is structured by `Application` and `RowNames`. For information about available log runs, see `list`.

`run_ids` — Simulation run ID numbers
vector of rows in available runs table

Identifies the simulation runs to delete from the target computer. The *`run_ids`* are rows in the available file logging data table. For information about available log runs, see `list`.

See Also

Enable File Log | File Log | `Target` | `abort` | `disable` | `enable` | `import` | `list`

Topics
“Signal Logging Basics”

Introduced in R2020b

list

Package: slrealtime

Get information about available file logs of signal data

Syntax

```
run_info = list(target_object.FileLog)
```

Description

`run_info = list(target_object.FileLog)` gets information about file log data that is available for the real-time applications on the target computer.

When a real-time application stops on a target computer that is connected to Simulink Real-Time, the target computer uploads file log data to the development computer. If the target computer is not connected when the application stops, the file logging data for applications accumulates on the target computer. The `list` function returns a table that lists the accumulated file logging data for application runs.

Examples

Get Available File Log Information for Applications

For target computer object `tg`, get information about available file log data for installed applications.

```
my_run_info = list(tg.FileLog)
```

```
my_run_info =

  3×3 table

          Application            StartDate           Size
        ________________    ____________________    _____

    1.  "slrt_ex_osc_rt_t"  12-Dec-2019 21:59:31    94944
    2.  "slrt_ex_osc_rt_t"  12-Dec-2019 21:59:45    84736
    3.  "slrt_ex_osc"       12-Dec-2019 21:59:57    82176
```

Import file log data from application runs 1 and 2 in the available file logs table.

```
import(tg.FileLog,1:2);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`run_info` — Structure of information about file log runs

struct

The *`run_info`* structure is a MATLAB table that is structured by `Application` and `RowNames`.

See Also

Enable File Log | File Log | `Target` | `abort` | `disable` | `discard` | `enable` | `import`

Topics

“Signal Logging Basics”

Introduced in R2020b

import

Package: `slrealtime`

Import file log data from target computer

Syntax

```
import(target_object.FileLog,'app_name')
import(target_object.FileLog,run_info)
import(target_object.FileLog,run_ids)
```

Description

`import(target_object.FileLog,'app_name')` imports file log signal data from available simulation runs for the selected real-time application.

As the function imports available file logging data, the function deletes the data from the target computer. For information about the availability of file logging data, see `list`.

`import(target_object.FileLog,run_info)` imports file log signal data for the selected table of available simulation runs. To create the table, use the `list` function.

`import(target_object.FileLog,run_ids)` imports file log signal data for the selected simulation runs.

If a Simulink Real-Time model has File Log blocks, when you load the real-time application on the target computer, file logging is enabled. This default operation is the same as enabling file logging by using the command `enable`.

To control file logging with the Enable File Log block, when you load the real-time application on the target computer, disable file logging by using the command `disable`.

When the development computer is connected to the target computer and the real-time application stops, the file log data is uploaded to the Simulation Data Inspector. For a standalone target computer that does file logging when not connected, after connecting the development and target computers, upload the file logging data for the application.

Note: When the Simulink Real-Time imports file log data from the target computer and uploads the data to the Simulation Data Inspector, the data is deleted from the target computer. This data is deleted whether the data upload occurs when the real-time application stops for a connected target computer or when you use the `import` function for a standalone (disconnected) target computer. File log data for imported runs of the application is deleted.

Examples

Import File Log Data for Application

For target computer object `tg` with simulation run data available for real-time application `my_app`, import file log data to the Simulation Data Inspector for the application.

```
import(tg.FileLog,'app_name')
```

Import File Log Data for Applications Runs

For target computer object `tg` with simulation run data available for real-time applications, get available simulation run information, and then import file log data.

Get table of available simulation run information. Import file log data from applications runs to the Simulation Data Inspector.

```
my_run_info = list(tg.FileLog);
import(tg.FileLog,my_run_info);
```

Alternatively, you can get the available file log information and import the file log data in one step.

```
import(tg.FileLog,tg.FileLog.list);
```

Import File Log Data for Selected Application Runs

For target computer object `tg` with simulation run data available for real-time applications `slrt_ex_osc_rt_t` and `slrt_ex_osc`, import file log data to the Simulation Data Inspector for selected simulation runs. For more information, see `list`.

Get table of available simulation run information.

```
my_run_info = list(tg.FileLog)
```

```
my_run_info =

  3×3 table

           Application              StartDate           Size 
         __________________    ____________________    _____

    1.   "slrt_ex_osc_rt_t"    12-Dec-2019 21:59:31    94944
    2.   "slrt_ex_osc_rt_t"    12-Dec-2019 21:59:45    84736
    3.   "slrt_ex_osc"         12-Dec-2019 21:59:57    82176
```

Import file log data from application runs 1 and 2 in the available file logs table.

```
import(tg.FileLog,1:2);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

app_name — Real-time application name
character vector | string scalar

Provides name of real-time application MLDATX file that you built from the model.

Example: `'slrt_ex_osc'`

`run_info` — Structure of information about file log runs

struct

The *`run_info`* structure is a MATLAB table that is structured by `Application` and `RowNames`. For information about available log runs, see `list`.

`run_ids` — Simulation run ID numbers

vector of rows in available runs table

Identifies the simulation runs to import from the target computer into the Simulation Data Inspector. The *`run_ids`* are rows in the available file logging data table. For information about available log runs, see `list`.

See Also

Enable File Log | File Log | `Target` | `abort` | `disable` | `discard` | `enable` | `list`

Topics

"Signal Logging Basics"

Introduced in R2020b

Target.ptpd

Target Computer PTP Daemon

Description

A Target.ptpd object represents the RTOS PTP daemon that runs on a target computer and provides access to methods and properties related to the PTP daemon.

The object provides access to methods and properties that:

- Start and stop the PTP daemon.
- Configure the PTP daemon startup command.
- Enable auto start of the PTP daemon.
- Retrieve status information about the PTP daemon.

Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name if the characters you type are unique for the property.

Creation

A Target.ptpd object is created when you create a Target object by using the slrealtime command. After you create and connect to the Target object, you can access the Target.ptpd object. This example creates and connects to Target object tg, and then starts the PTP daemon on the target computer.

```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
```

Properties

AutoStart — Enable PTP daemon start on target computer start
0 (off) (default) | 1 (on)

When AutoStart is enabled, after the target computer boots, the RTOS PTP daemon starts by using the command specified in the Target.ptpd object Command property.

Example: 0

Command — Specify the PTP daemon start command
'ptpd -L -K -g' (default) | character vector

The default value for the Command property is a command string that starts the RTOS PTP daemon with enable multiple daemons (-L), devctl() support (-K), and slave (-g). To change from slave to master, stop the PTP daemon, change the command string, and start the PTP daemon. To enable hardware time stamp and achieve best master-slave clock synchronization, bind the PTP daemon to an Ethernet i210 interface by using the -b switch. For more information about PTP commands, see the QNX Neutrino documentation.

Example: 'ptpd -L -K -g'

Object Functions

start	Start the PTP daemon on the target computer
stop	Stop the PTP daemon on the target computer
status	View the PTP daemon status on the target computer

Examples

Configure PTP Daemon Properties

The Target.ptpd.Command and Target.ptpd.AutoStart properties configure operation of the PTP daemon.

Create a Target object and connect to the target computer. Creating a Target object creates a child Target.ptpd object. Connecting to the target computer provides access to the Target.ptpd object.

```
tg = slrealtime('TargetPC1');
connect(tg);
```

View the Target.ptpd object Command property value.

```
tg.ptpd.Command
```

```
ans =

    'ptpd -L -K -g'
```

View the Target.ptpd object AutoStart property value.

```
tg.ptpd.AutoStart
```

```
ans =

  logical

   0
```

Configure Target.ptpd object Command property value for master and AutoStart property value for auto start.

```
stop(tg.ptpd); % ensure that the daemon is stopped
tg.ptpd.Command = 'ptpd -L -K -G';
tg.ptpd.AutoStart = 1;
start(tg.ptpd); % start daemon with new values
```

See Also

IEEE 1588 Read Parameter | start | status | stop

Topics

“Precision Time Protocol”
“PTP Prerequisites”

Introduced in R2020b

start

Package: slrealtime

Start the PTP daemon on the target computer

Syntax

start(target_object.ptpd)

Description

start(target_object.ptpd) starts the RTOS PTP daemon on the target computer

Examples

Start PTP Daemon

The start command starts the PTP daemon on the target computer by running the command selected in the Target.ptpd object Command property value.

Create a Target object and connect to the target computer. Creating a Target object creates a child Target.ptpd object. Start the PTP daemon on the target computer.

```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

IEEE 1588 Read Parameter | status | stop

Topics
“Precision Time Protocol”
“PTP Prerequisites”

Introduced in R2020b

stop

Package: slrealtime

Stop the PTP daemon on the target computer

Syntax

```
stop(target_object.ptpd)
```

Description

stop(target_object.ptpd) stops the RTOS PTP daemon on the target computer.

Examples

Stop PTP Daemon

The stop command stops the PTP daemon on the target computer.

Create a Target object and connect to the target computer. Creating a Target object creates a child Target.ptpd object. Start the PTP daemon on the target computer. Run the real-time application. Stop the PTP daemon.

```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
% ... run real-time application
stop(tg.ptpd);
```

Input Arguments

target_object — Object that represent target computer
slrealtime.Target object

Provides access to methods that manipulate the target computer properties.

Example: tg

See Also

IEEE 1588 Read Parameter | start | status

Topics
“Precision Time Protocol”
“PTP Prerequisites”

Introduced in R2020b

status

Package: slrealtime

View the PTP daemon status on the target computer

Syntax

```
status(target_object.ptpd)
```

Description

`status(target_object.ptpd)` displays the status of the PTP daemon on the target computer.

Examples

View PTP Daemon Status

The `status` command displays status of the PTP daemon on the target computer. This status includes PTP clock synchronization information.

Create a `Target` object and connect to the target computer. Creating a Target object creates a child `Target.ptpd` object. Start the PTP daemon on the target computer. View status of the PTP daemon.

```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
status(tg.ptpd)
```

```
ans = 

  struct with fields:

             Running: 1
              Devctl: 1
               Error: ''
    OffsetFromMaster: 0
       MasterToSlave: 0
       SlaveToMaster: 0
         OneWayDelay: 0
        SavedOptions: [1×1 struct]
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

See Also

IEEE 1588 Read Parameter | `start` | `stop`

Topics

"Precision Time Protocol"
"PTP Prerequisites"

Introduced in R2020b

Targets

Configure and manage target objects

Description

A `Targets` object represents target computers that are defined on the development computer and provides access to methods related to the target computers.

Creation

`targets_object = slrealtime.Targets()` constructs a `Targets` object representing target computers that are connected to the development computer.

Example: “Create Targets Object, Add Target Computers, Set IP Address” on page 1-92

Object Functions

addTarget	Add target computer definition to targets object
removeTarget	Remove target computer definition from targets object
getTargetSettings	Get target computer environment settings
getDefaultTargetName	Get default target computer name
setDefaultTargetName	Set default target computer name

Examples

Create Targets Object, Add Target Computers, Set IP Address

To work with multiple target computers, make the computer names available by using a targets object.

Create targets object `my_tgs`. Add target computers to the targets object. Assign target computers to target objects. Create target settings object and list the target computer names.

```
my_tgs = slrealtime.Targets();
% do not need to add default target 'TargetPC1'
addTarget(my_tgs,'TargetPC2');
addTarget(my_tgs,'TargetPC3');

% assign target computers to target objects
tg1 = slrealtime('TargetPC1');
tg2 = slrealtime('TargetPC2');
tg3 = slrealtime('TargetPC3');

% list target computer names
my_tgs_settings = getTargetSettings(my_tgs);
my_tgs_settings.name
```

```
ans =
    'TargetPC1'


ans =

    'TargetPC2'
```

Set `Target` object `tg2` IP address to `'10.10.10.25'` by using the `targetSettings` property.

```
tg1.targetSettings.address = '10.10.10.25';
tg1.targetSettings;
```

To set the IP address on the target computer, use the `setipaddr` function.

See Also

`addTarget` | `getTargetSettings` | `removeTarget`

Introduced in R2020b

addTarget

Package: slrealtime

Add target computer definition to targets object

Syntax

```
addTarget(targets_object,target_name)
```

Description

addTarget(targets_object,target_name) adds the definition for a target computer, represented by the name target_name. Do not add or remove the default target computer name TargetPC1.

Examples

Add Target 'TargetPC2' to System

Add target computer definition 'TargetPC2' to Targets object my_tgs.

```
my_tgs = slrealtime.Targets();
addTarget(my_tgs,'TargetPC2');
```

Input Arguments

targets_object — Object that represents target computers
Targets object

Provides access to methods that manipulate the target computers and their target settings.

Example: tgs

Data Types: struct

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

Example: 'TargetPC1'

Data Types: char | string

See Also

Targets | getTargetSettings | removeTarget

Introduced in R2020b

getTargetSettings

Package: `slrealtime`

Get target computer environment settings

Syntax

```
settings_object = getTargetSettings(targets_object)
```

Description

`settings_object = getTargetSettings(targets_object)` gets the environment settings for the target computers that are connected to the development computer.

Examples

Create Targets Object and View Settings

Create `Targets` object `my_tgs`. Get target settings for object.

```
my_tgs = slrealtime.Targets();
my_tgs_settings = getTargetSettings(my_tgs)
```

```
my_tgs_settings = 

  TargetSettings with properties:

            name: 'TargetPC1'
         address: '10.10.10.35'
         sshPort: 22
         xcpPort: 5555
        username: 'slrt'
    userPassword: 'slrt'
    rootPassword: 'root'
```

Get target computer name properties from `Targets` object.

```
my_tgs_settings.name
```

```
ans =

    'TargetPC1'


ans =

    'TargetPC2'
```

Get target computer address properties from `Targets` object.

```
my_tgs_settings.address
```

```
ans =

    '10.10.10.15'


ans =

    '10.10.10.25'
```

To change target computer settings, use the properties of the `Target` object.

Input Arguments

`targets_object` — Object that represents target computers
`Targets` object

Provides access to methods that manipulate the target computers and their target settings.

Example: `tgs`

Data Types: `struct`

Output Arguments

`settings_object` — Settings object that represents target computer settings
`slrealtime.TargetSettings` object

Object containing target computer environment settings.

Data Types: `struct`

See Also

`Targets` | `addTarget` | `removeTarget`

Introduced in R2020b

removeTarget

Package: slrealtime

Remove target computer definition from targets object

Syntax

```
removeTarget(targets_object,target_name)
```

Description

removeTarget(targets_object,target_name) removes the definition and settings for the target computer represented by target_name from the target_object. The target objects associated with that target_name become invalid. Do not add or remove the default target computer name TargetPC1.

Examples

Remove Target 'TargetPC2' from System

Remove target computer definition 'TargetPC2' from Targets object my_tgs.

```
removeTarget(my_tgs,'TargetPC2')
```

Input Arguments

targets_object — Object that represents target computers
Targets object

Provides access to methods that manipulate the target computers and their target settings.

Example: tgs

Data Types: struct

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

Example: 'TargetPC1'

Data Types: char | string

See Also
Targets | addTarget | getTargetSettings

Introduced in R2020b

getDefaultTargetName

Package: `slrealtime`

Get default target computer name

Syntax

```
getDefaultTargetName(targets_object,target_name)
```

Description

`getDefaultTargetName(targets_object,target_name)` gets the name of the default target computer.

Examples

Get Default Target Computer Name

Create `Targets` object `my_tgs`. Get default target computer name.

```
my_tgs = slrealtime.Targets();
getDefaultTargetName(my_tgs)
```

```
ans =

    'TargetPC1'
```

Input Arguments

`targets_object` — Object that represents target computers
`Targets` object

Provides access to methods that manipulate the target computers and their target settings.

Example: `tgs`

Data Types: `struct`

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

Example: `'TargetPC1'`

Data Types: `char` | `string`

See Also

`Targets` | `addTarget` | `removeTarget` | `setDefaultTargetName`

Introduced in R2020b

setDefaultTargetName

Package: `slrealtime`

Set default target computer name

Syntax

`setDefaultTargetName(targets_object,target_name)`

Description

`setDefaultTargetName(targets_object,target_name)` sets the name for the default target computer.

Examples

Set Default Target Computer Name

Create `Targets` object `my_tgs`. Set default target computer name.

```
my_tgs = slrealtime.Targets();
setDefaultTargetName(my_tgs,'TargetPC1')
```

Input Arguments

`targets_object` — Object that represents target computers
`Targets` object

Provides access to methods that manipulate the target computers and their target settings.

Example: `tgs`

Data Types: `struct`

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

Example: `'TargetPC1'`

Data Types: `char` | `string`

See Also

`Targets` | `addTarget` | `getDefaultTargetName` | `removeTarget`

Introduced in R2020b

Application

Represent application files on development computer

Description

An application object represents application files on the development computer. You can create application objects for real-time applications that you build from models.

An application object provides access to methods and properties that let you work with the application blocks and signals.

Creation

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

The `slrealtime.Application` function accepts these arguments:

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

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

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

Create an application object for real-time application `slrt_ex_osc_inport`.

```
app_object = slrealtime.Application('slrt_ex_osc_inport');
```

Example: “Extract ASAP2 File” on page 1-101

Example: “Update Root-Level Inport Data” on page 1-102

Example: “Get and Set Application Options” on page 1-102

Example: “Get Application Signals and Parameters” on page 1-103

Properties

`ApplicationName` — Name of real-time application
character vector | string scalar

This property is read-only.

Name of real-time application created when you built the application.

`ModelName` — Name of Simulink model

character vector | string scalar

This property is read-only.

Name of the Simulink model from which you build the real-time application.

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

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

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

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

`Options` — Real-time application options

character vector | string scalar

This property is read-only.

Use the Options property to get and set real-time application options. For a usage example, see “Get and Set Application Options” on page 1-102. The options are:

- `fileLogMaxRuns` selects the number of simulation runs that are stored for the real-time application when file logging is enabled.
- `loglevel` selects the log message level for the target computer system log. The available levels are `error`, `warning`, `info`, `debug`, and `trace`.
- `pollingThreshold` selects the sample rate below which the RTOS thread scheduler switches polling mode —instead of interrupt-driven mode— for clocking the real-time application. Polling mode can be useful for reducing sample time jitter. But, enabling this option causes the real-time application to consume a CPU core completely to clock and execute the base rate.
- `stoptime` selects the stop time for the real-time application.

Object Functions

extractASAP2	Extract generated A2L file from real-time application file
getInformation	Get real-time application information
getParameters	Get real-time application parameters
getSignals	Get real-time application signals
updateRootLevelInportData	Replace external input data in real-time application with input data

Examples

Extract ASAP2 File

Retrieve ASAP2 file from real-time application.

Create application object for the real-time application.

```
app_obj =  slrealtime.Application("myModel.mldatx");
```

Retrieve ASAP2 file from the real-time application.

```
extractASAP2(app_obj);
```

Update Root-Level Inport Data

Change waveform data from square wave to sine wave.

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

```
waveform = sinewave;
```

Create an application object.

```
app_object = slrealtime.Application('slrt_ex_osc_inport');
```

Update inport data.

```
updateRootLevelInportData(app_object)
```

Download the updated inport data to the default target computer.

```
tg = slrealtime('TargetPC1');
load(tg, 'slrt_ex_osc_inport');
```

Get and Set Application Options

You can get and set real-time application options by using the application `Options` property.

Create an application object.

```
my_app = slrealtime.Application('slrt_ex_osc_inport');
```

View application options by getting the application Options property values.

```
my_app.Options.get
```

```
ans =

  struct with fields:

      fileLogMaxRuns: 1
            loglevel: "info"
    pollingThreshold: 1.0000e-04
            stoptime: Inf
```

Change the application stop time value option.

```
my_app.Options.set("stoptime",20);
```

Save application options to a MATLAB variable. Apply options from variable to the real-time application by using the load function.

```
my_options = my_app.Options.get;
save("my_options.mat", "my_options");
```

```
load("my_options.mat", "my_options");
my_app.Options.set(my_options);
```

Get Application Signals and Parameters

You can get real-time application signals and parameters by using the `getParameters` and `getSignals` functions.

Create an application object.

```
my_app = slrealtime.Application('slrt_ex_param_tuning')
```

```
my_app = 

  Application with properties:

    ApplicationName: 'slrt_ex_param_tuning'
          ModelName: 'slrt_ex_param_tuning'
           UserData: []
            Options: [1×1 slrealtime.internal.ApplicationOptions]
```

Get the application Signals values as structures in an array.

```
my_sigs = getSignals(my_app)
```

```
my_sigs = 

  1×9 struct array with fields:

    BlockPath
    PortIndex
    SignalLabel
```

View application signals as array elements.

```
my_sigs(1).BlockPath
```

```
ans = 

    'slrt_ex_param_tuning/Gain'
```

Get the application Parameters values as structures in an array.

```
my_params = getParameters(my_app)
```

```
my_params = 

  1×7 struct array with fields:

    BlockPath
    BlockParameterName
```

View application parameters as array elements.

```
my_params(1).BlockParameterName
```

```
ans =

    'Gain'
```

See Also

extractASAP2 | getInformation | getParameters | getSignals | updateRootLevelInportData

Topics

"Define and Update Inport Data"
"Define and Update Inport Data by Using MATLAB Language"

Introduced in R2020b

extractASAP2

Extract generated A2L file from real-time application file

Syntax

```
extractASAP2(app_obj)
extractASAP2(app_obj,Name,Value)
```

Description

`extractASAP2(app_obj)` retrieves an A2L file from a real-time application file and save the file in the working folder.

`extractASAP2(app_obj,Name,Value)` specifies additional options to retrieve an A2L file by using one or more Name, Value pair arguments. For example, you can specify a location for saving the A2L file. You can provide the target IP address to update it in A2L file before saving it.

Examples

Extract A2L File Generated

Retrieve the A2L file from real-time application.

```
% extract a2l file from mymodel application file
app_obj = slrealtime.Application('mymodel.mldatx')
extractASAP2(app_obj)
```

Extract A2L File and Save with Custom Name

Retrieve the A2L file from real-time application and then save the A2L file with the custom name specified.

```
% save extracted a2l file with custom name
app_obj = slrealtime.Application('mymodel.mldatx')
extractASAP2(app_obj,'FileName','MyApp')
```

Extract A2L File and Save in Custom Location

Retrieve the A2L file from real-time application and then save the A2L file in the specified location.

```
% save extracted a2l file in custom location
app_obj = slrealtime.Application('mymodel.mldatx')
extractASAP2(app_obj,'Folder','C:\workspace')
```

Extract A2L File and Update The Target IP Address

Retrieve the A2L file from real-time application and update the target IP Address.

```
% save extracted a2l file by updating IP Address
app_obj = slrealtime.Application('mymodel.mldatx')
extractASAP2(app_obj,'TargetIPAddress','192.168.1.1')
```

Input Arguments

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

object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'FileName', 'CustomName'`, `'Folder', 'C:\workspace'`

FileName — Custom name to save the A2L file

character vector | string scalar

Save the A2L file retrieved from the real-time application with custom name specified.

Example: `'FileName'`, `'MyModel'`

Folder — Folder location to save A2L file

character vector | string scalar

Full path of the folder in which to save the A2L file.

Example: `'Folder'`, `'D:\SLRT\Applications'`

TargetIPAddress — Custom target IP address to be used in A2L file

character vector | string scalar

Extract the A2L file from the real-time application by updating the target IP address.

Example: `'TargetIPAddress'`, `'192.168.1.1'`

See Also

`Application` | `updateRootLevelInportData`

Introduced in R2020b

getInformation

Package: `slrealtime`

Get real-time application information

Syntax

```
info_struct = getInformation(app_object)
```

Description

`info_struct = getInformation(app_object)` gets the application Information values as a structure with properties. Use the `getInformation` function to get real-time application and model information from the Application object.

Examples

Get Application Information

You can get real-time application information by using the `getInformation` function.

Create an application object.

```
my_app = slrealtime.Application('slrt_ex_osc_inlined')
```

```
my_app = 

  Application with properties:

    ApplicationName: 'slrt_ex_osc_inlined'
          ModelName: 'slrt_ex_osc_inlined'
           UserData: []
            Options: [1×1 slrealtime.internal.ApplicationOptions]
```

Get the application Information values as a structure with properties.

```
my_app_info = getInformation(my_app)
```

```
my_app_info = 

  struct with fields:

                ApplicationName: 'slrt_ex_osc_inlined'
        ApplicationCreationDate: '2020-04-21 10:29:08'
    ApplicationLastModifiedDate: '2020-04-21 10:29:10'
                      ModelName: 'slrt_ex_osc_inlined'
                   ModelVersion: '1.22'
              ModelCreationDate: '1999-07-16 09:55:35'
          ModelLastModifiedDate: '2020-04-13 16:10:31'
            ModelLastModifiedBy: 'The MathWorks, Inc.'
                ModelSolverType: 'Fixed-step'
                ModelSolverName: 'ode4'
                  MatlabVersion: '9.9.0.1343993 (R2020b) Prerelease'
```

View application information values as array elements.

```
my_app_info.ApplicationCreationDate
```

```
ans =

    '2020-04-21 10:29:08'
```

Input Arguments

app_object — Object that represents real-time application files on the development computer
object

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

Output Arguments

info_struct — Information values as a structure with properties
a structure with properties

The Information values are read-only. The structures in the array are:

- ApplicationName — real-time application name
- ApplicationCreationDate — real-time application creation date
- ApplicationLastModifiedDate — real-time application modified date
- ModelName — name of model from which real-time application was built
- ModelVersion — model version
- ModelCreationDate — model creation date
- ModelLastModifiedDate — model modified date
- ModelLastModifiedBy — model modified by
- ModelSolverType — model solver type
- ModelSolverName — model solver name
- MatlabVersion — MATLAB version

See Also

Application | Target | getSignals

Topics

“Add App Designer App to Inverted Pendulum Model”

Introduced in R2020b

getParameters

Package: slrealtime

Get real-time application parameters

Syntax

```
params_struct = getParameters(app_object)
```

Description

params_struct = getParameters(app_object) gets the application Parameters values as structures in an array. Use the getParameters function to get tunable parameter information from the Application object.

Examples

Get Application Parameters

You can get real-time application parameters by using the getParameters function.

Create an application object.

```
my_app = slrealtime.Application('slrt_ex_param_tuning')
```

```
my_app = 

  Application with properties:

    ApplicationName: 'slrt_ex_param_tuning'
          ModelName: 'slrt_ex_param_tuning'
           UserData: []
            Options: [1×1 slrealtime.internal.ApplicationOptions]
```

Get the application Parameters values as structures in an array.

```
my_params = getParameters(my_app)
```

```
my_params = 

  1×7 struct array with fields:

    BlockPath
    BlockParameterName
```

View application parameter values as array elements.

```
my_params(1).BlockParameterName
```

```
ans =

    'Gain'
```

Input Arguments

`app_object` — Object that represents real-time application files on the development computer

object

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

Output Arguments

`params_struct` — Parameters values as structures in an array

structures in an array

The Parameters values are read-only. The structures in the array are:

- `BlockPath` — block path of the parameter in the application
- `BlockParameterName` — block parameter name in the application

See Also

`Application` | `Target` | `getSignals`

Topics

"Add App Designer App to Inverted Pendulum Model"

Introduced in R2020b

getSignals

Package: slrealtime

Get real-time application signals

Syntax

```
sigs_struct = getSignals(app_object)
```

Description

sigs_struct = getSignals(app_object) gets the application Signals values as structures in an array. Use the getSignals function to get signal information for signals that are marked for streaming to the Simulation Data Inspector from the Application object.

Examples

Get Application Signals

You can get real-time application signals by using the getSignals function.

Create an application object.

```
my_app = slrealtime.Application('slrt_ex_param_tuning')
```

```
my_app = 

  Application with properties:

    ApplicationName: 'slrt_ex_param_tuning'
          ModelName: 'slrt_ex_param_tuning'
           UserData: []
            Options: [1×1 slrealtime.internal.ApplicationOptions]
```

Get the application Signals values as structures in an array.

```
my_sigs = getSignals(my_app)
```

```
my_sigs = 

  1×9 struct array with fields:

    BlockPath
    PortIndex
    SignalLabel
```

View application signals as array elements.

```
my_sigs(1).BlockPath
```

```
ans =

    'slrt_ex_param_tuning/Gain'
```

Input Arguments

`app_object` — Object that represents real-time application files on the development computer
object

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

Output Arguments

`sigs_struct` — Signals values as structures in an array
structures in an array

The Signals values are read-only. The structures in the array are:

- `BlockPath` — block path of the signal in the application
- `PortIndex` — port index of the signal in the application
- `SignalLabel` — label of the signal in the application

See Also

`Application` | `Target` | `getParameters`

Topics
"Add App Designer App to Inverted Pendulum Model"

Introduced in R2020b

updateRootLevelInportData

Package: `slrealtime`

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

Syntax

```
updateRootLevelInportData(app_object)
```

Description

`updateRootLevelInportData(app_object)` replaces external input data in a real-time application with new input data.

Examples

Update Inport Data with Application Object

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

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

```
waveform = sinewave;
```

Create an application object.

```
app_object = slrealtime.Application('slrt_ex_slrt_osc_inport');
```

Update inport data.

```
updateRootLevelInportData(app_object)
```

Download the updated inport data to the default target computer.

```
tg = slrealtime('TargetPC1');
load(tg, 'slrt_ex_osc_inport');
```

Input Arguments

`app_object` — Object that represents real-time application files on the development computer
object

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

See Also

`Application` | `Target`

Topics
“Define and Update Inport Data”
“Define and Update Inport Data by Using MATLAB Language”

Introduced in R2020b

SystemLog

Get current console log from target computer

Description

A SystemLog object represents the console log from the target computer at the time the object is created by using the slrealtime.SystemLog function.

Creation

slog_object = slrealtime.SystemLog(target_object) creates a system log object that contains a table of current target computer console messages in its messages property.

To view the target computer console log, you can create a SystemLog object and view its messages property or use the Simulink Real-Time system log viewer slrtLogViewer.

Properties

messages — table of current console log messages
table of messages

The messages property value is a table of the current console log messages.

Object Functions

slrtLogViewer   Open Simulink Real-Time System Log Viewer tab in Simulink Real-Time Explorer to view the console log from target computer

Examples

Create and View System Log

To work with multiple target computers, make the computer names available by using a targets object.

Create targets object my_tgs. Add target computers to the targets object. Assign target computers to target objects. Create target settings object and list the target computer names.

```
tg = slrealtime('TargetPC1');
slog = slrealtime.SystemLog(tg);
slog.messages
```

```
ans =

  13×4 table

         Timestamp                              Message                              Severity    Catego
    ____________________    _____________________________________________________    ________    ______
    26-Nov-2019 21:27:33    "Target IP address: 10.10.10.35"                   "info"        2
    26-Nov-2019 21:28:44    "Loading model slrt_ex_ExecutionProfAndConc"       "info"        0
    26-Nov-2019 21:28:44    "Loading model slrt_ex_ExecutionProfAndConc"       "info"        0
    26-Nov-2019 21:28:44    "Waiting for start command"                        "info"        0
    26-Nov-2019 21:28:44    "Waiting for start command"                        "info"        0
    26-Nov-2019 21:28:44    "loglevel = info"                                  "info"        0
    26-Nov-2019 21:28:44    "loglevel = info"                                  "info"        0
    26-Nov-2019 21:28:44    "pollingThreshold = 0.0001"                        "info"        0
    26-Nov-2019 21:28:44    "pollingThreshold = 0.0001"                        "info"        0
    26-Nov-2019 21:28:44    "relativeTimer = [unset]"                          "info"        0
    26-Nov-2019 21:28:44    "relativeTimer = [unset]"                          "info"        0
    26-Nov-2019 21:28:44    "stoptime = 2"                                     "info"        0
    26-Nov-2019 21:28:44    "stoptime = 2"
```

See Also

slrtLogViewer

Introduced in R2020b

Instrument

Create real-time instrument object

Description

An `slrealtime.Instrument` object streams signal data from a real-time simulation running on a target computer to a development computer.

Creation

`instrument_object = slrealtime.Instrument('appName')` creates an empty instrument object for an existing real-time application `appName`.

Example: "Create Instrument Object for Real-Time Application" on page 1-118

`instrument_object = slrealtime.Instrument()` creates an empty instrument object without an assigned real-time application.

Example: "Create Instrument Object without Real-Time Application" on page 1-118

Properties

`AxesTimeSpan` — Axes time span in seconds
`Inf` (default) | double

The `AxesTimeSpan` property controls the time axis (x-axis) for all axes in an App Designer UI. When set to `Inf`, the signal value from the real-time application running on the target computer is displayed in the axes. If you change to a value, for example 10, the time axis for all axes is set to that value, for example 10 seconds.

`AxesTimeSpanOverrun` — Axes time span overrun response
`scroll` (default) | `wrap`

The `AxesTimeSpanOverrun` property controls the response for axes in an App Designer UI when the data overruns the `AxesTimeSpan` property value. When the `AxesTimeSpan` property value is `Inf`, the `AxesTimeSpanOverrun` property has no effect. When the `AxesTimeSpan` property value is set in seconds, the time axis for all axes is set to a finite width (time range). When a signal value from the real-time application exceeds the largest time value on the x-axis, the axes can either **scroll** or **wrap**.

`Application` — Name of real-time application
character vector | string

You can set the value of the `Application` property to an existing real-time application when you create the Instrument object or you can set the value later. After value is written to this property, it become read-only. You can not change the Application property value directly after creating the object. The property value can only be changed after object creation by using the `validate` function.

Object Functions

addInstrumentedSignals	Find instrumented signals and add these to real-time instrument object
addSignal	Add signal for streaming to be available in callback
clearScalarAndLineData	Clear data from children of real-time instrument object
connectCallback	Add callback that responds to new data
connectLine	Connect signal for streaming to axes
connectScalar	Add signal for streaming to scalar display
delete	Delete real-time instrument object
generateScript	Generate script that creates scalar and axes controls from signals, scalars, and lines in real-time instrument object
getCallbackDataForSignal	Get callback data for a signal in real-time instrument object
removeCallback	Removed callback from real-time instrument object
removeSignal	Remove signal from real-time instrument object
validate	Validate signals in instrument object

Examples

Create Instrument Object for Real-Time Application

- Create instrument object *hInst* for an existing real-time application *appName*.

```
appName = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(appName);
```

Create Instrument Object without Real-Time Application

- Create instrument object *hInst* without assigning a real-time application. This approach is useful when building a GUI and the real-time application MLDATX file is not available.

```
hInst = slrealtime.Instrument();
```

Apply Instrument Object Methods

- This example shows how to create an Instrument object, apply Instrument object methods, and remove the object.

```
inst = slrealtime.Instrument();

inst.connectScalar(app.Numeric1, 'ScalarDouble1');
inst.connectScalar(app.Gauge1,   'ScalarDouble1');
inst.connectScalar(app.Numeric2, "ScalarDouble2");
inst.connectScalar(app.Gauge2,   "ScalarDouble2");

inst.connectScalar(app.Text1, "myString", 'Callback', @(t,d)string(d));
inst.connectScalar(app.Text2, "myString", 'Callback', @(t,d)string(d), 'Decimation', 2);

inst.connectScalar(app.Lamp0, "TrafficLight", 'PropertyName', 'Visible', 'Callback', @(t,d)st
inst.connectScalar(app.Lamp1, "TrafficLight", 'PropertyName', 'Visible', 'Callback', @(t,d)st
inst.connectScalar(app.Lamp2, "TrafficLight", 'PropertyName', 'Visible', 'Callback', @(t,d)st

ls2 = slrealtime.instrument.LineStyle();
```

```
ls2.Marker = '*';
ls2.MarkerSize = 4;
ls2.Color = 'black';
inst.connectLine(app.Axes1, "SineWave", 'ArrayIndex', 5, 'LineStyle', ls2, 'Callback', @(t,d)
inst.connectLine(app.Axes1, "SineWave");

inst.connectCallback(@(o,e)customPlot(o,e,app)); % plot sine waves added together with amplit

tg=slrealtime;
tg.addInstrument(inst);

inst.AxesTimeSpan = 10;

inst.AxesTimeSpanOverrun = 'wrap';

inst.AxesTimeSpan = Inf;

tg.removeInstrument(inst);
```

See Also

addInstrumentedSignals | addSignal | clearScalarAndLineData | connectCallback | connectLine | connectScalar | delete | generateScript | getCallbackDataForSignal | removeCallback | removeSignal | validate

Topics

“Instrumentation Apps for Real-Time Applications”

Introduced in R2020b

addInstrumentedSignals

Package: slrealtime

Find instrumented signals and add these to real-time instrument object

Syntax

```
addInstrumentedSignals(instrument_object)
```

Description

addInstrumentedSignals(instrument_object) finds real-time application signals that are marked for streaming to the Simulation Data Inspector and adds these instrumented signals to the real-time instrument object. If the instrument_object does not have an assigned real-time application MLDATX file, the addSignal command issues an error message.

Examples

Add Instrumented Signals to Instrument Object

Select real-time application file. Create instrument object. Add instrumented signals to the instrument object.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
addInstrumentedSignals(hInst);
```

Input Arguments

instrument_object — Object that represents real-time instrument
object

To create the instrument object, use the Instrument function.

Example: hInst

See Also

Instrument | addSignal | clearScalarAndLineData | connectCallback | connectLine | connectScalar | delete | generateScript | getCallbackDataForSignal | removeCallback | removeSignal | validate

Introduced in R2020b

addSignal

Package: `slrealtime`

Add signal for streaming to be available in callback

Syntax

```
addSignal(instrument_object,blockPath,portIndex,Name,Value)
addSignal(instrument_object,signalName,Name,Value)
```

Description

`addSignal(instrument_object,blockPath,portIndex,Name,Value)` adds a signal by using the block path and the port index for streaming to make the signal available in a callback. Use this approach when you do not use the signal in a scalar displace or line plot.

`addSignal(instrument_object,signalName,Name,Value)` adds a signal by using the signal name for streaming to make the signal available in a callback. Use this approach when you do not use the signal in a scalar displace or line plot.

Examples

Add Signal by Using Block Path and Port Index

Add a signal for streaming to the real-time instrument object by using the block path and port index.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
addSignal(hInst,'slrt_ex_tank/ControlValue',1);
```

Add Signal by Using Signal Name

Add a signal for streaming to the real-time instrument object by using the signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
addSignal(hInst,'ControlValueOut');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`blockPath` — Block path for block with signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `slrt_ex_tank/ControlValue`

`portIndex` — Index of block port that is connected to signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: 1

`signalName` — Name of signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: `ControlValueOut`

`Name,Value` — Name-value pairs that set properties values
name-value pair

The *`Name,Value`* pair argument selects the signal properties that are added to the instrument object *`instrument_object`* and sets values for the properties.

Example: `'Decimation',2`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Decimation',2`

`BusElement` — Nonvirtual bus element
signal name (character vector)

Specifies a particular element of a nonvirtual bus to stream. The syntax for the `BusElement` value:

- Starts with the selected index for Array of Buses `'(index).'` or empty for scalar bus signals
- Contains the path from the first level down to the leaf element
- Separates each level of the hierarchy with a period `'.'`
- Has a leaf as last level
- Expresses the index for Array of Buses in the path as `'(index)'`

Example: `'BusElement','u1'`

Example: `'BusElement','u4(1).b'`

Example: `'BusElement','(1).a'`

Decimation — Decimation value

1 (default) | numeric, scalar, positive value

Specifies a decimation value for the signal.

Example: 'Decimation',2

See Also

Instrument | addInstrumentedSignals | clearScalarAndLineData | connectCallback | connectLine | connectScalar | delete | generateScript | getCallbackDataForSignal | removeCallback | removeSignal | validate

Introduced in R2020b

clearScalarAndLineData

Package: slrealtime

Clear data from children of real-time instrument object

Syntax

```
clearScalarAndLineData(instrument_object)
```

Description

`clearScalarAndLineData(instrument_object)` clears data from a real-time instrument object. For each scalar and axes control connected through `connectLine` or `connectScalar`, the `clearScalarAndLineData` function clears the UI control data. In a gauge for example, the Value field is reset and the needle points to 0. On axes for example, the line data is cleared and the axes are empty.

Examples

Clear Data from Instrument Object

Select real-time application file. Create instrument object. Clear data from instrument object.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
% . . . hInst streams data
clearScalarAndLineData(hInst);
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Introduced in R2020b

connectCallback

Package: slrealtime

Add callback that responds to new data

Syntax

connectCallback(instrument_object,hCallback)

Description

connectCallback(instrument_object,hCallback) adds a callback that responds to new data, which is available from the target computer. The eventData for the callback shares all the new data available from the target computer since the last time the callback was executed.

Examples

Add Callback for Available New Data

Add a callback that responds to new data available from the target computer and stream that data to the real-time instrument object.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
```

Input Arguments

instrument_object — Object that represents real-time instrument
object

To create the instrument object, use the Instrument function.

Example: hInst

hCallback — MATLAB function handle evaluated when new data is available
object

The callback responds to new data becoming available for streaming.

Example: @my_callback

See Also

Instrument | addInstrumentedSignals | addSignal | clearScalarAndLineData | connectLine | connectScalar | delete | generateScript | getCallbackDataForSignal | removeCallback | removeSignal | validate

Introduced in R2020b

connectLine

Package: slrealtime

Connect signal for streaming to axes

Syntax

```
connectLine(instrument_object,hAxis,blockPath,portIndex,Name,Value)
connectLine(instrument_object,hAxis,signalName,Name,Value)
```

Description

`connectLine(instrument_object,hAxis,blockPath,portIndex,Name,Value)` connects a signal by using the block path and port index for streaming to axes.

`connectLine(instrument_object,hAxis,signalName,Name,Value)` connects a signal by using a signal name for streaming to axes.

Examples

Connect Signal by Block Path and Port Index

Connect a signal for streaming to the real-time instrument object and axes object by using the block path and port index.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
connectLine(hInst,myAxis,'slrt_ex_tank/ControlValue',1);
```

Connect Signal by Signal Name

Connect a signal for streaming to the real-time instrument object and axis object by using a signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
connectLine(hInst,myAxis,'ControlValueOut');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`hAxis` — Handle to axis of a figure or UI figure

object

To create an axes object, use `hAxis = gca` or `hAxis = axes ()`.

Example: `myAxes`

`blockPath` — Block path for block with signal connected to one of its outports

character vector

For the selected block, `gcb` returns the full block path name.

Example: `slrt_ex_tank/ControlValue`

`portIndex` — Index of block port that is connected to signal for streaming

integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: 1

`signalName` — Name of signal for streaming

character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: `ControlValueOut`

`Name,Value` — Pair that set properties values

name-value pair

The *`Name,Value`* pair argument selects the signal properties that are added to the instrument object *`instrument_object`* and sets values for the properties.

Example: `'Decimation',2`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Decimation',2`

`ArrayIndex` — Array index of multi-element signal

integer

Selects an element of a multi-element signal.

Example: `'ArrayIndex',5`

`BusElement` — Nonvirtual bus element

signal name (character vector)

Specifies a particular element of a nonvirtual bus to stream. The syntax for the `BusElement` value:

- Starts with the selected index for Array of Buses `'(index).'` or empty for scalar bus signals
- Contains the path from the first level down to the leaf element
- Separates each level of the hierarchy with a period `'.'`
- Has a leaf as last level
- Expresses the index for Array of Buses in the path as `'(index)'`

Example: `'BusElement','u1'`

Example: `'BusElement','u4(1).b'`

Example: `'BusElement','(1).a'`

`Callback` — Function handle

function handle

Provides function handle for accepting (time,data) arguments and returning data.

Example: `'Callback', @(t,d)(d+app.Offset.Value)`

`Decimation` — Decimation value

`1` (default) | numeric, scalar, positive value

Specifies a decimation value for the signal.

Example: `'Decimation',2`

`LineStyle` — LineStyle object selection

`'none'` (default) | `'-'` | `'--'` | `':'` | `'-.'`

A `slrealtime.LineStyle` object that customizes the line appearance. Valid values to select the object are `'-'`, `'--'`, `':'`, `'-.'`, or `'none'`.

Example: `'LineStyle', '-'`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Introduced in R2020b

connectScalar

Package: `slrealtime`

Add signal for streaming to scalar display

Syntax

```
connectScalar(instrument_object,hDisplay,blockPath,portIndex,Name,Value)
connectScalar(instrument_object,hDisplay,signalName,Name,Value)
```

Description

`connectScalar(instrument_object,hDisplay,blockPath,portIndex,Name,Value)` connects a signal by using the block path and port index for streaming to a scalar display as a scalar object.

`connectScalar(instrument_object,hDisplay,signalName,Name,Value)` connects a signal by using a signal name for streaming to a scalar display as a scalar object.

Examples

Connect Signal by Using Block Path and Port Index

Connect a signal for streaming to the real-time instrument object and display the object by using the block path and port index.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
connectScalar(hInst,myDisplay,'slrt_ex_tank/ControlValue',1);
```

Connect Signal by Using Signal Name

Connect a signal for streaming to the real-time instrument object and display the object by using a signal name.

```
% added signal name to model before building mldatxfile
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
connectScalar(hInst,myDisplay,'ControlValueOut');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument

object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`hDisplay` — Handle to a scalar display
object

The scalar display object displays the streaming data from the instrument in an edit box, gauge, or other display. object.

Example: `myGauge`

`blockPath` — Block path for block with signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `slrt_ex_tank/ControlValue`

`portIndex` — Index of block port that is connected to signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: 1

`signalName` — Name of signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: `ControlValueOut`

`Name,Value` — Pair that set properties values
name-value pair

The *`Name,Value`* pair argument selects the signal properties that are added to the instrument object *`instrument_object`* and sets values for the properties.

Example: `'Decimation',2`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Decimation',2`

`ArrayIndex` — Array index of multi-element signal
integer

Selects an element of a multi-element signal.

Example: `'ArrayIndex',5`

`BusElement` — Nonvirtual bus element
signal name (character vector)

Specifies a particular element of a nonvirtual bus to stream. The syntax for the `BusElement` value:

- Starts with the selected index for Array of Buses `'(index).'` or empty for scalar bus signals
- Contains the path from the first level down to the leaf element
- Separates each level of the hierarchy with a period `'.'`
- Has a leaf as last level
- Expresses the index for Array of Buses in the path as `'(index)'`

Example: `'BusElement','u1'`

Example: `'BusElement','u4(1).b'`

Example: `'BusElement','(1).a'`

`Callback` — Function handle

function handle

Provides function handle for accepting (time,data) arguments and returning data.

Example: `'Callback', @(t,d)(d+app.Offset.Value)`

`Decimation` — Decimation value

`1` (default) | numeric, scalar, positive value

Specifies a decimation value for the signal.

Example: `'Decimation',2`

`LineStyle` — LineStyle object selection

`'none'` (default) | `'-'` | `'--'` | `':'` | `'-.'`

A `slrealtime.LineStyle` object that customizes the line appearance. Valid values to select the object are `'-'`, `'--'`, `':'`, `'-.'`, or `'none'`.

Example: `'LineStyle', '-'`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Introduced in R2020b

delete

Package: slrealtime

Delete real-time instrument object

Syntax

delete(instrument_object)

Description

delete(instrument_object) deletes a real-time instrument object.

Examples

Delete Instrument Object

Delete instrument object hInst. If the instrument object is streaming data from a real-time application, stop streaming and delete the instrument object.

```
% previously . . .
% . . . created a target object
% . . . loaded/started an application on target
% . . . created an instrument object
% . . . optionally streamed data by using instrument object
delete(hInst)
```

Input Arguments

instrument_object — Object that represents real-time instrument
object

To create the instrument object, use the Instrument function.

Example: hInst

See Also

Instrument | addInstrumentedSignals | addSignal | clearScalarAndLineData | connectCallback | connectLine | connectScalar | generateScript | getCallbackDataForSignal | removeCallback | removeSignal | validate

Introduced in R2020b

generateScript

Package: slrealtime

Generate script that creates scalar and axes controls from signals, scalars, and lines in real-time instrument object

Syntax

```
generateScript(instrument_object)
```

Description

`generateScript(instrument_object)` generates an M-script that creates scalar and axes controls from the signals, scalars, and lines in a real-time instrument object.

Examples

Generate Script from Instrument Object

Select real-time application file. Create instrument object. Generate script that creates scalar and axes controls from instrument object.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
generateScript(hInst);
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Introduced in R2020b

getCallbackDataForSignal

Package: `slrealtime`

Get callback data for a signal in real-time instrument object

Syntax

```
[time,data] = getCallbackDataForSignal(instrument_object,blockPath,portIndex,
Name,Value)
[time,data] = getCallbackDataForSignal(instrument_object,signalName)
```

Description

`[time,data] = getCallbackDataForSignal(instrument_object,blockPath,portIndex,Name,Value)` gets callback data from the target computer for a signal by using the block path and the port index.

`[time,data] = getCallbackDataForSignal(instrument_object,signalName)` gets callback data from the target computer for a signal by using the signal name. The `eventData` for the callback shares all the new data available from the target computer since the last time the callback was executed.

Examples

Get Callback Data by Using Block Path and Port Index

Get callback data for a signal by using the block path and port index of the signal in the real-time application file.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
addSignal(hInst,'slrt_ex_tank/ControlValue',1);
% . . . hInst streams data
[cv_time,cv_data] = getCallbackDataForSignal(hInst,'slrt_ex_tank/ControlValue',1);
```

Get Callback Data by Using Signal Name

Get callback data for a signal by using the signal name of the signal in the real-time application file.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
addSignal(hInst,'ControlValue');
```

```
% . . . hInst streams data
[cv_time,cv_data] = getCallbackDataForSignal(hInst,'ControlValue');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument

object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`blockPath` — Block path for block with signal connected to one of its outports

character vector

For the selected block, `gcb` returns the full block path name.

Example: `slrt_ex_tank/ControlValue`

`portIndex` — Index of block port that is connected to signal for streaming

integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: 1

`signalName` — Name of signal for streaming

character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: `ControlValueOut`

Output Arguments

`time` — Time data from target computer

time data

The time value is the current time returned from the target computer.

`data` — Signal data from target computer

signal data

The data value is the current signal data returned from the target computer.

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `removeCallback` | `removeSignal` | `validate`

Introduced in R2020b

removeCallback

Package: slrealtime

Removed callback from real-time instrument object

Syntax

```
removeCallback(instrument_object,hCallback)
```

Description

`removeCallback(instrument_object,hCallback)` removes a callback from a real-time instrument object.

Examples

Remove Callback Data from Instrument Object

Remove callback from instrument object.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
% . . . hInst streams data
removeCallback(hInst,@my_callback);
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`hCallback` — MATLAB function handle evaluated when new data is available
object

The callback stops responding to new data available for streaming.

Example: `@my_callback`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeSignal` | `validate`

Introduced in R2020b

removeSignal

Package: slrealtime

Remove signal from real-time instrument object

Syntax

```
removeSignal(instrument_object,blockPath,portIndex,Name,Value)
removeSignal(instrument_object,signalName,Name,Value)
```

Description

`removeSignal(instrument_object,blockPath,portIndex,Name,Value)` removes a signal from a real-time instrument object by using the block path and the port index.

`removeSignal(instrument_object,signalName,Name,Value)` removes a signal from a real-time instrument object.

Examples

Remove Signal by Using Block Path and Port Index

Remove a signal from the real-time instrument object by using the block path and port index.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
addSignal(hInst,'slrt_ex_tank/ControlValue',1);
% . . . hInst streams data
removeSignal(hInst,'slrt_ex_tank/ControlValue',1);
```

Remove Signal by Using Signal Name

Remove a signal from the real-time instrument object by using the signal name.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
addSignal(hInst,'ControlValueOut');
% . . . hInst streams data
removeSignal(hInst,'ControlValueOut');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument
object

To create the instrument object, use the `Instrument` function.

Example: `hInst`

`blockPath` — Block path for block with signal connected to one of its outports
character vector

For the selected block, `gcb` returns the full block path name.

Example: `slrt_ex_tank/ControlValue`

`portIndex` — Index of block port that is connected to signal for streaming
integer

For the selected signal, the output port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: 1

`signalName` — Name of signal for streaming
character vector

For the selected signal, the port index and signal name are visible in the signal hierarchy available in Simulink Real-Time explorer or in the Model Data Editor.

Example: `ControlValueOut`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `validate`

Introduced in R2020b

validate

Package: slrealtime

Validate signals in instrument object

Syntax

```
instrument_object = validate(instrument_object,rtApplication)
```

Description

instrument_object = validate(instrument_object,rtApplication) validates the instrument object against the signals present in the real-time application. The validate operation outputs the list of signals that are present in the instrument object, but are not available in the real-time application.

Examples

Validate Instrument Object

For input instrument object mySignals that contains named signals Integ_out, Integ1_out, and Integ2_out, check whether the named signals are available in real-time application slrt_ex_osc. Any unavailable signals are added to the output instrument object unavailSignals.

```
unavailSignals = validate(mySignals,'slrt_ex_osc')
```

```
Integ2_out
```

Input Arguments

instrument_object — Select instrument object
object

The input *instrument_object* argument identifies the object to validate. To create an instrument object, use the Instrument function.

Example: hInst

rtApplication — Select real-time application for instrument
rtApplicationName

The *rtApplicationName* argument identifies the real-time application that contains the signals listed in the input instrument object. The validation identifies any signals in the input instrument object that are not available in the real-time application.

Example: slrt_ex_osc

Output Arguments

`instrument_object` — Select instrument object
slrealtime.Instrument object

The output *`instrument_object`* argument identifies the object for validation information.

Example: `hInst`

See Also

`Instrument` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal`

Introduced in R2020b

ProfilerData

Data returned from profiler

Description

Internal format returned by profiler and displayed by using public functions.

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

- To display the profile data for a section of the model, click the membrane button next to the report section.
- To display the TET data for the section in the Simulation Data Inspector, click the plot time series data button .
- To view the section in Simulink Editor, click the link next to the expand tree button [+].
- To view the lines of generated code corresponding to the section, click the expand tree button [+], and then click the view source button .

The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The Code Execution Profiling Report lists the model sections. The numbers underneath the bars indicate the processor cores.

Creation

`getProfilerData`

Object Functions

plot	Generate execution profiler plot
report	Generate profiler report

Examples

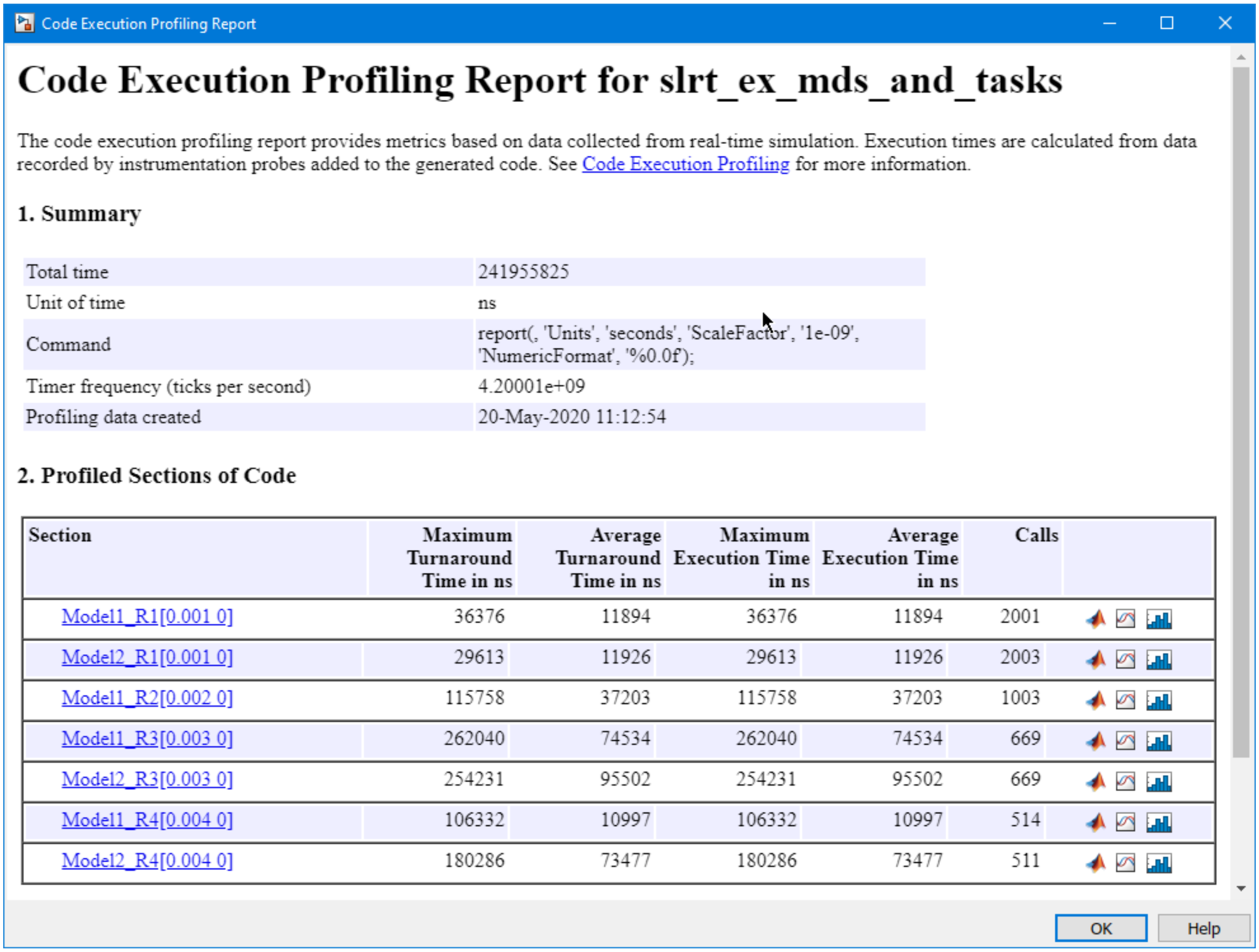
Run Profiler and Explicitly Display Profiler Data

Load the application. Start the profiler. Start the application. Stop the profiler. Retrieve profile execution data. Call `report` and `plot` on the data.

```
tg = slrealtime('TargetPC1');
rtwbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);
start(tg);

stopProfiler(tg);
stop(tg);

profiler_object = getProfilerData(tg);
```

```
rocessing data on target computer, please wait ...
Transferring data from target computer to host computer, please wait ...
Processing data on host computer, please wait ...

Code execution profiling data for model slrt_ex_mds_and_tasks.
```

```
report(profiler_object);
```



Code Execution Profiling Report for slrt_ex_mds_and_tasks

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

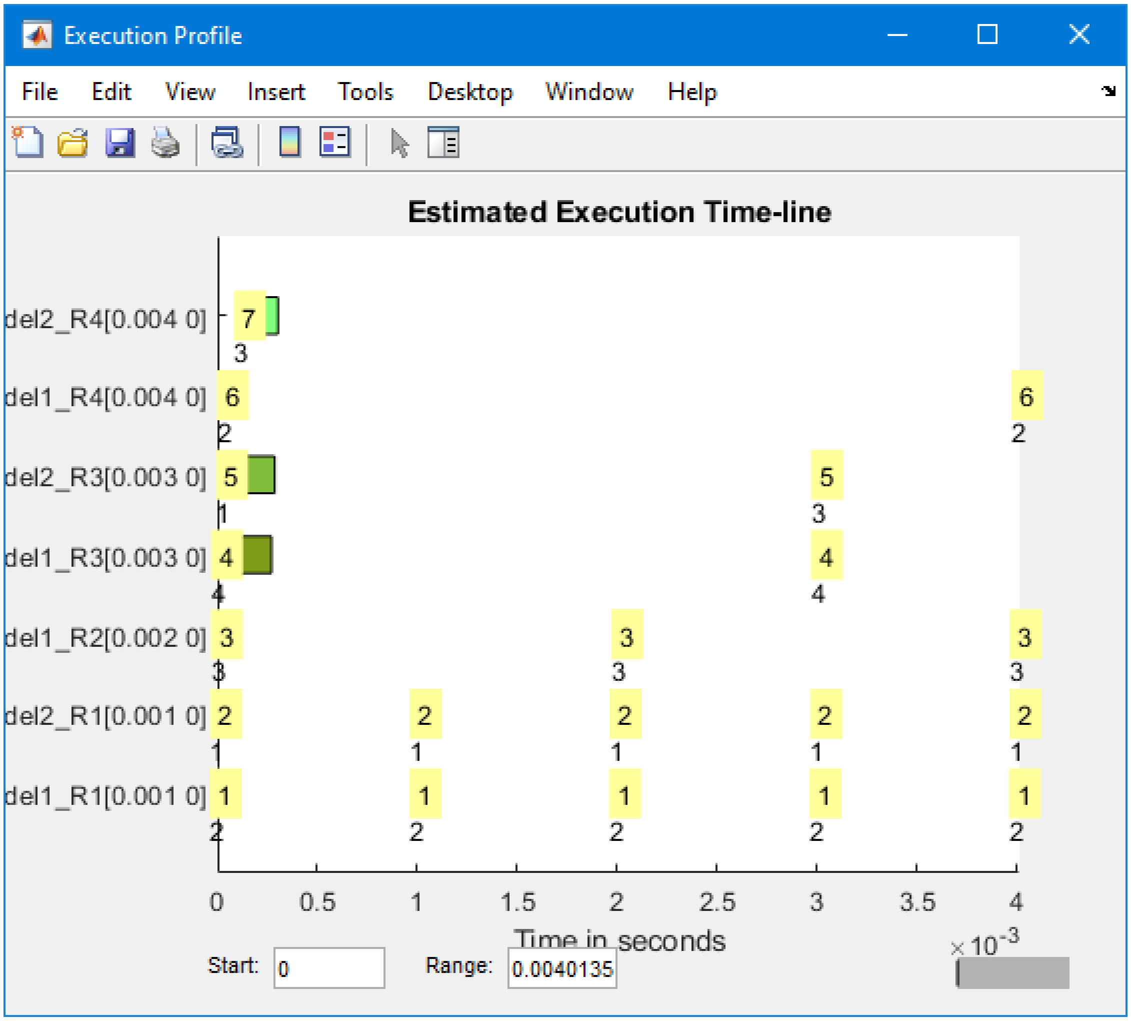
1. Summary

Total time	241955825
Unit of time	ns
Command	report(, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	4.20001e+09
Profiling data created	20-May-2020 11:12:54

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls	
Model1_R1[0.001 0]	36376	11894	36376	11894	2001	
Model2_R1[0.001 0]	29613	11926	29613	11926	2003	
Model1_R2[0.002 0]	115758	37203	115758	37203	1003	
Model1_R3[0.003 0]	262040	74534	262040	74534	669	
Model2_R3[0.003 0]	254231	95502	254231	95502	669	
Model1_R4[0.004 0]	106332	10997	106332	10997	514	
Model2_R4[0.004 0]	180286	73477	180286	73477	511	

```
plot(profiler_object);
```



Start: 0

Range: 0.0040135

See Also

Enable Profiler | getProfilerData | plot | report | resetProfiler | startProfiler | stopProfiler

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

plot

Package: `slrealtime`

Generate execution profiler plot

Syntax

```
plot(profiler_object)
```

Description

`plot(profiler_object)` generates a plot from the profiler data.

The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The Code Execution Profiling Report lists the model sections. The numbers underneath the bars indicate the processor cores.

Examples

Run Profiler and Plot Profiler Data

The real-time application is already loaded. Start the profiler. Start the application.

```
tg = slrealtime('TargetPC1');
startProfiler(tg);
start(tg);
```

Stop the profiler. Stop the application.

```
stopProfiler(tg);
stop(tg);
```

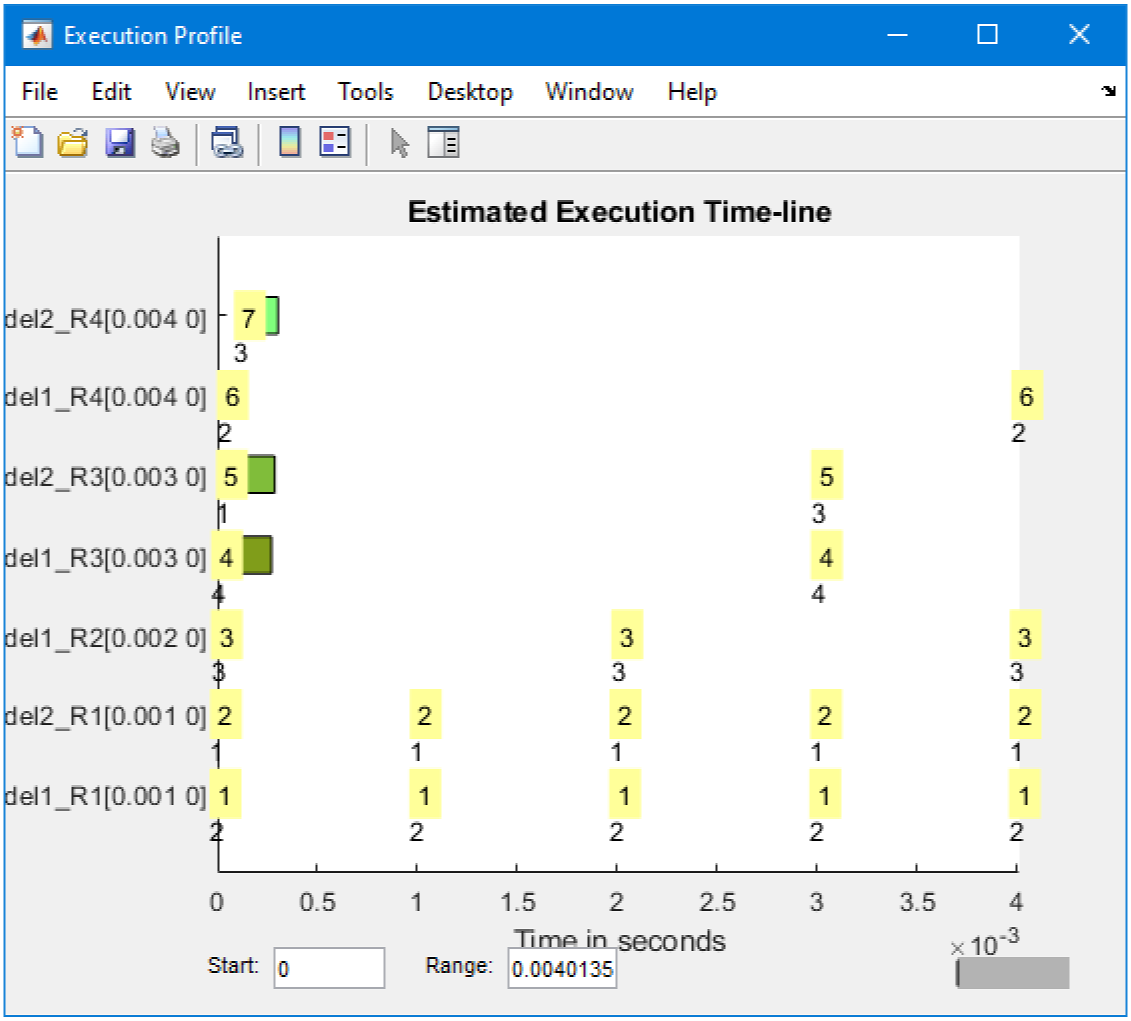
Retrieve profiler data.

```
profiler_object = getProfilerData(tg);
```

```
Processing data, please wait ...
```

Call `plot` function on the data.

```
plot(profiler_object);
```



Input Arguments

`profiler_object` — Object that contains profiler result

structure

MATLAB variable that you can use to access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has these fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters** dialog box. In the **Real-Time** tab, click **Hardware Settings**. Select the **Code Generation** > **Verification** > **Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

`ProfilerData` | `getProfilerData` | `report`

Topics

“Execution Profiling for Real-Time Applications”

Introduced in R2020b

report

Package: `slrealtime`

Generate profiler report

Syntax

```
report(profiler_object)
```

Description

`report(profiler_object)` generates a report from the profiler data.

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

- To display the profile data for a section of the model, click the membrane button next to the section.
- To display the TET data for the section in the Simulation Data Inspector, click the plot time series data button.
- To view the section in Simulink Editor, click the link next to the expand tree button [+].
- To view the lines of generated code corresponding to the section, click the expand tree button [+], and then click the view source button.

Examples

Run Profiler and Report Profiler Data

The real-time application is already loaded. Start the profiler. Start the application.

```
tg = slrealtime('TargetPC1');
startProfiler(tg);
start(tg);
```

Stop the profiler. Stop the application.

```
stopProfiler(tg);
stop(tg);
```

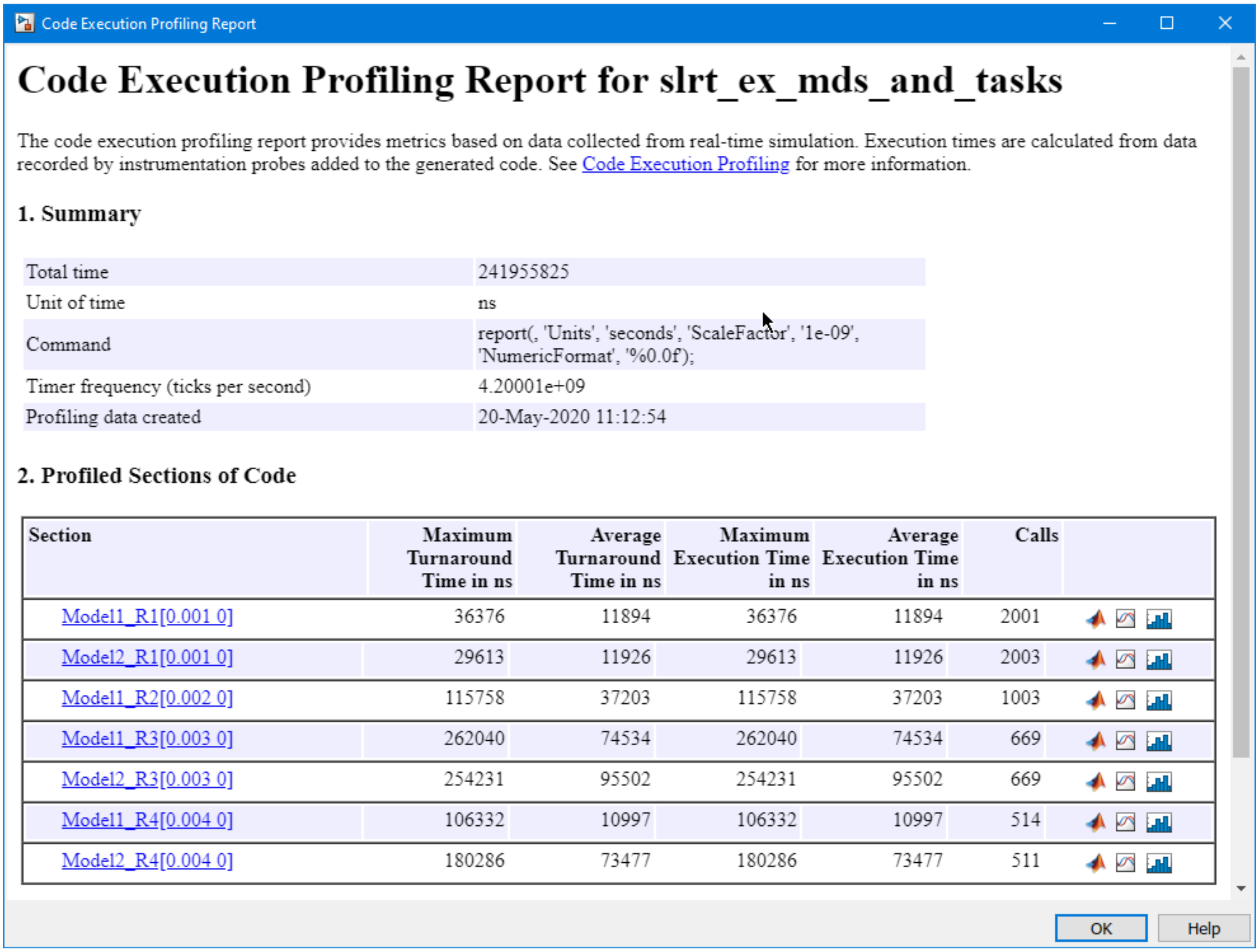
Retrieves profiler data.

```
profiler_object = getProfilerData(tg);
```

```
Processing data, please wait ...
```

Call the `report` function on the results data.

```
report(profiler_object);
```



Code Execution Profiling Report for slrt_ex_mds_and_tasks

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

Total time	241955825
Unit of time	ns
Command	report(, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	4.20001e+09
Profiling data created	20-May-2020 11:12:54

2. Profiled Sections of Code

Section	Maximum Turnaround Time in ns	Average Turnaround Time in ns	Maximum Execution Time in ns	Average Execution Time in ns	Calls
Model1_R1[0.001 0]	36376	11894	36376	11894	2001
Model2_R1[0.001 0]	29613	11926	29613	11926	2003
Model1_R2[0.002 0]	115758	37203	115758	37203	1003
Model1_R3[0.003 0]	262040	74534	262040	74534	669
Model2_R3[0.003 0]	254231	95502	254231	95502	669
Model1_R4[0.004 0]	106332	10997	106332	10997	514
Model2_R4[0.004 0]	180286	73477	180286	73477	511

Input Arguments

`profiler_object` — Object that contains profiler result

structure

MATLAB variable that you can use to access the result of the profiler execution. You display the profiler data by calling the `plot` and `report` functions.

The structure has these fields:

- `TargetName` — Name of target computer in target computer settings.
- `ModelInfo` — Information about model on which profiler ran:
  - `ModelName` — Name of real-time application.
  - `MATLABRelease` — MATLAB release under which model was built.

You can access the data in the *`profiler_object`* variable. To access the profiler data, before running the profiler, open the **Configuration Parameters** dialog box. In the **Real-Time** tab, click

Hardware Settings. Select the **Code Generation** > **Verification** > **Workspace variable** option and set the value to `executionProfile`. Select the **Save options** option and set the value to `All data`. After running the profiler, use the technique described for the `Sections` function.

See Also

`ProfilerData` | `getProfilerData` | `plot`

Topics

"Execution Profiling for Real-Time Applications"

Introduced in R2020b

slrealtime.etherCAT.filterNotifications

Package: `slrealtime`

Display EtherCAT notifications in human-readable format

Syntax

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

Description

`slrealtime.etherCAT.filterNotifications()` prints the valid notification values and their text descriptions.

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

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

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

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

`[filtered_values suppressed_values] = slrealtime.etherCAT.filterNotifications(tlog, olog, suppress)` returns a structure vector containing the filtered values and a structure containing a summary of the suppressed values.

Examples

Print Valid Notifications

Print the valid notification values and their text descriptions

```
slrealtime.etherCAT.filterNotifications
```

```
slrealtime.EtherCAT.filterNotifications
(     1): State changed
(     2): Cable connected
(     3): Scanbus finished
(     4): Distributed clocks initialized
(     5): DC slave synchronization deviation received
(     8): DCL initialized
(     9): DCM inSync
(    21): Successful slave state transition.
(   100): Queue raw command response notification
( 65537): Cyclic command: Working count error
( 65538): Master init command: Working count error
( 65539): Slave init command: Working count error
( 65540): EOE mbox receive: Working count error (deprecated)
( 65541): COE mbox receive: Working count error (deprecated)
( 65542): FOE mbox receive: Working count error (deprecated)
( 65543): EOE mbox send: Working count error
( 65544): COE mbox send: Working count error
( 65545): FOE mbox send: Working count error
( 65546): Frame response error: No response
( 65547): Slave init command: No response
( 65548): Master init command: No response
( 65550): Timeout when waiting for mailbox init command response
( 65551): Cyclic command: Not all slaves in op state
( 65552): Ethernet link (cable) not connected
( 65554): Redundancy: Line break detected
( 65555): Cyclic command: A slave is in error state
( 65556): Slave error status change
( 65557): Station address lost (or slave missing) - FPRD to ...
          AL_STATUS failed
( 65558): SOE mbox receive: Working count error (deprecated)
( 65559): SOE mbox send: Working count error
( 65560): SOE mbox write responded with an error
( 65561): COE mbox SDO abort
( 65562): Client registration dropped, possibly call to ...
          ecatConfigureMaster by other thread (RAS)
( 65563): Redundancy: Line is repaired
( 65564): FOE mbox abort
( 65565): Invalid mail box data received
( 65566): PDI watchdog expired on slave, thrown by IST
( 65567): Slave not supported (if redundancy is activated and ...
          slave doesn't fully support autoclose
( 65568): Slave in unexpected state
( 65569): All slave devices are in operational state
( 65570): VOE mbox send: Working count error
( 65571): EEPROM checksum error detected
( 65572): Crossed lines detected
( 65573): Junction redundancy change
(196610): ScanBus mismatch
(196611): ScanBus mismatch. A duplicate HC group was detected
(262146): HC enhance detect all groups done
(262147): HC probe all groups done
(262148): HC topology change done
(262149): Slave disappears
(262150): Slave appears
```

Get Time and Data Log from EtherCAT Get Notifications Block

Export time log and data log for a simulation run from the Simulation Data Inspector. Apply the `slrealtime.etherCAT.filterNotification` command to the log data.

In this example, the output of the EtherCAT Get Notifications block connects to a File Log block. After the simulation run stops, Simulink Real-Time uploads the file log data to the Simulation Data Inspector. You can use the `slrealtime.etherCAT.filterNotification` command on the log data.

In your model, connect the output of the EtherCAT Get Notifications block connects to a File Log block.

Build the model, and then download and run the real-time application.

Open the Simulation Data Inspector.

While the real-time application is running, the Simulation Data Inspector lists any signals that are marked for logging, for example as `Run 1:<modelname>@TargetPC1`. When model execution stops, the Simulation Data Inspector moves that run to the archive. Then, Simulink Real-Time uploads the signal data from the File Log block to the Simulation Data Inspector. This data appears, for example as `Run 2:<modelname>@TargetPC1[FileLog][Current]`.

To apply use the `slrealtime.etherCAT.filterNotification` command on the log data, export the whole data set as a single data set to the MATLAB workspace. These steps create a 1x1 data set that contains the variable notifications.

a In the Simulation Data Inspector, right-click the `Run 2:` line.
b Select **Export Data ...**. That opens a dialog.
c For **Export:**, select **Selected runs and signals**.
d For **To:**, select **Base workspace** and provide a variable name for the export, such as `notifications`.

To get the `timelog` and the `datalog` use:

```
timelog = notifications{1}.Values.Time;
datalog = notifications{1}.Values.Data;
```

To print notifications from normal operations, run the `filterNotifications` command with this data:

```
slrealtime.EtherCAT.filterNotifications(timelog, datalog, [])
```

```
 Time      Code     Description
0.040000 (     3) Scanbus finished
0.045000 (     1) State changed
1.199000 (     4) Distributed clocks initialized
1.202000 (     1) State changed
4.198000 (     9) DCM inSync
4.200000 (     5) DC slave synchronization deviation received
4.350000 (     1) State changed
4.357000 (     1) State changed
```

Return Filtered Notifications from Normal Operation

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

There are cases in which message filtering or suppression is useful. In certain error situations, you may see many notifications about one particular situation that can hide other significant notifications.

This situation could be a large number of working count errors or frame response errors, for example, that hide other notifications that you may need to identify how to recover from the situation.

For information about creating the `timelog` and `datalog` variables, see “Get Time and Data Log from EtherCAT Get Notifications Block” on page 1-151.

```
[filtered_values suppressed_values] = ...
    slrealtime.etherCAT.filterNotifications(timelog, datalog, [1])
```

```
 Time      Code     Description
0.040000 (     3) Scanbus finished
1.199000 (     4) Distributed clocks initialized
4.198000 (     9) DCM inSync
4.200000 (     5) DC slave synchronization deviation received

Suppressed notifications:

      1: 4 times [0.045000 : 4.357000]
State changed
```

Input Arguments

`tlog` — Time log on target computer

vector

Use exported time log data from signal data displayed in the Simulation Data Inspector. See Get Time and Data Log from EtherCAT Get Notifications Block on page 1-151 .

Example: `timelog`

Data Types: `double`

`olog` — Output log on target computer

matrix

Use exported data log data from signal data displayed in the Simulation Data Inspector. See Get Time and Data Log from EtherCAT Get Notifications Block on page 1-151 .

Example: `outputlog`

Data Types: `double`

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

vector

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

Example: `65546`

Example: []

Data Types: `double`

Output Arguments

`filtered_values` — Return filtered values as structure vector

vector

Each element of filtered_values is a structure containing:

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

suppressed_values — Return suppressed codes as structure vector
vector

Each element of suppressed_values is a structure containing:

- val (double) — Notify code
- first (double) — Timestamp of first occurrence
- last (double) — Timestamp of last occurrence
- count (double) — Number of instances found

Tips

- Common error conditions, such as an unplugged Ethernet cable, can cause thousands of unwanted notifications that hide useful notifications. To filter unwanted notifications, use the suppress vector.

See Also

EtherCAT Get Notifications

Introduced in R2020b

slrealtime.getSupportInfo

Creates `slrealtimeinfo.txt` file that provides information about Simulink Real-Time installation

Syntax

```
slrealtime.getSupportInfo
slrealtime.getSupportInfo(model_name)
```

Description

`slrealtime.getSupportInfo` creates an `slrealtimeinfo.txt` file that provides information about the Simulink Real-Time installation for MathWorks support.

`slrealtime.getSupportInfo(model_name)` creates an `slrealtimeinfo.txt` file that provides information about the Simulink Real-Time installation and a `model_name_configset.m` file that provides information about the open model for MathWorks support.

Examples

Get Support Information for MathWorks Support

To get support information about the Simulink Real-Time installation and a Simulink Real-Time model, open the model and run the `slrealtime.getSupportInfo` command.

```
open_system('slrt_ex_osc');
slrealtime.getSupportInfo('slrt_ex_osc');
```

Input Arguments

`model_name` — Simulink Real-Time model name
character vector | string scalar

Provides name of Simulink Real-Time model from which you are building a real-time application.

Example: `'slrt_ex_osc'`

See Also

`slrealtime.getCrashStack`

Introduced in R2020b

slrealtime.getCrashStack

Downloads and decodes core files from target computer and opens these in MATLAB editor

Syntax

```
files = slrealtime.getCrashStack(target_object)
```

Description

`files = slrealtime.getCrashStack(target_object)` downloads and decodes core files from the target computer and opens these in the MATLAB editor. The decoded core files help you investigate issues that cause application crashes on the target computer.

Examples

Get Crash Stack from Target Computer

Create a Target object tg. Connect to the target computer. Get and open any crash stack information that is available on the target computer.

```
tg = slrealtime;
connect(tg);
my_files = slrealtime.getCrashStack(tg);
```

Input Arguments

`target_object` — Object that represent target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`files` — names of created crash stack files
cell array of character vectors

Holds file names created from downloaded and decoded core files.

See Also
`slrealtime.getSupportInfo`

Introduced in R2020b