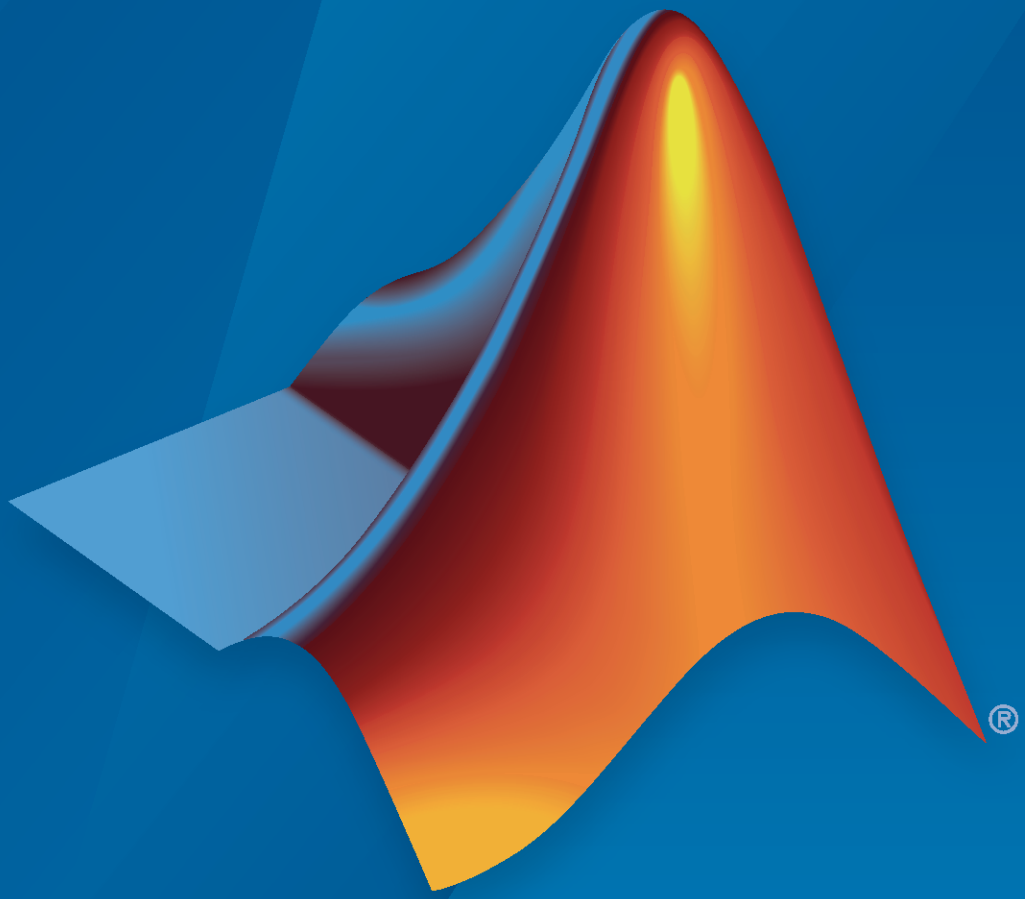
Simulink® Real-Time™

API Guide



MATLAB® & SIMULINK®

R2023a

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

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)
March 2021	Online only	Revised for Version 7.1 (Release 2021a)
September 2021	Online only	Revised for Version 7.2 (Release 2021b)
March 2022	Online only	Revised for Version 8.0 (Release 2022a)
September 2022	Online only	Revised for Version 8.1 (Release 2022b)
March 2023	Online only	Revised for Version 8.2 (Release 2023a)

Contents

MATLAB API

slrtExplorer

Package: `slrealtime`

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.

1. Open the Simulink Real-Time explorer. Type:

   ```
   slrtExplorer
   ```
2. To connect to the selected target computer, click **Connect**.
3. To select and load a real-time application, click **Load Application** and select the MLDATX file.
4. To select signals for streaming, click the application name, select signals from the **Signals** tab, and click the **Add selected signals** button.
5. To run the application and generate data for streaming, click the **Run** button.
6. 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.
7. To view the streaming signals, click the **Open in SDI** button.
8. After viewing the data, to stop the real-time application, click the **Stop** button.

Version History

Introduced in R2020b

See Also

`slrtLogViewer` | `slrtTETMonitor`

Topics
Simulink Real-Time Explorer

slrtLogViewer

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

Version History

Introduced in R2020b

See Also

`slrtExplorer` | `slrtTETMonitor` | `SystemLog`

Topics
Simulink Real-Time Explorer

slrtTETMonitor

Package: `slrealtime`

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.

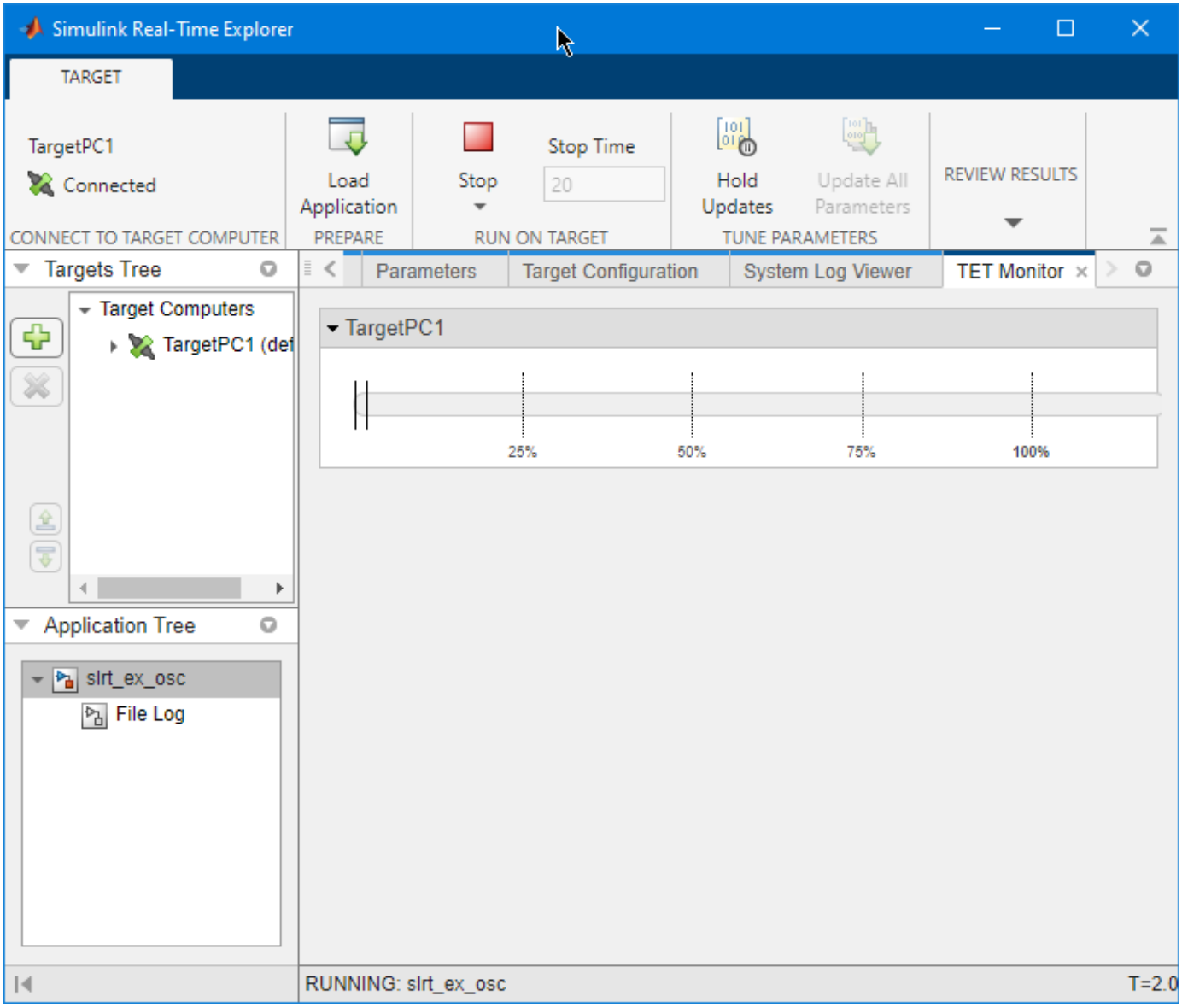
Examples

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.

1. Open the `slrt_ex_osc` model.
2. 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**.
3. Open the TET monitor. In the **Real-Time** tab, click **TET Monitor**. Or, in the Command Window, enter:

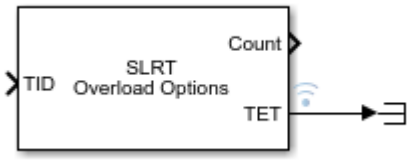
   ```
   slrtTETMonitor
   ```
4. When you run the real-time application, the TET monitor displays status.



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.

1. Open the `slrt_ex_osc` model.
2. Add a SLRT Overload Options block to the model.
3. In the block, set the **Enable TET Output** parameter value to `on`.
4. Select the TET output and mark it for data logging in the Simulation Data Inspector.



5. 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**.
6. Open the Simulation Data Inspector.
7. When you run the real-time application, the TET data is displayed in the Simulation Data Inspector.

Version History

Introduced in R2020b

See Also

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

Topics

"Data Logging with Simulation Data Inspector (SDI)"

Simulink Real-Time Explorer

slrtAppGenerator

Package: `slrealtime`

Generate instrument panel app to interact with target computer and real-time application running on target computer

Syntax

```
slrtAppGenerator
```

Description

`slrtAppGenerator` opens the Simulink Real-Time App Generator.

Simulink Real-Time app generator provides a UI for creating instrument panel apps for real-time applications running on the target computer. You can:

- Open a model file SLX or real-time application file MLDATX, and create an instrument panel app.
- Select signals and parameters to add to an instrument panel app.
- Configure controls for instrument panel app.
- Create instrument panel app.
- Save an App Generator session file MAT, and open it in a future App Generator session

Examples

Select Model or Real-Time Application and Create Instrument Panel App

To create an instrument panel app, you can open the App Generator from the Simulink Editor or open the App Generator from the MATLAB Command Window. If you open the App Generator from the MATLAB Command Window, you choose the model or real-time application for which you create the instrument panel app.

1. Open the Simulink Real-Time App Generator. Type:

   ```
   slrtAppGenerator
   ```
2. To select a model or real-time application, click the **New** button. Select the model or real-time application and click **Open**.
3. To create the instrument panel app, select signals and parameters to add to the instrument panel, configure the controls for your selections, and click the **Generate App** button.

   For more information about the App Generator, see **Simulink Real-Time App Generator**.

Version History

Introduced in R2022a

See Also

`slrtExplorer` | `slrtLogViewer` | `slrtTETMonitor`

Topics

Simulink Real-Time App Generator

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 1-15

Example: “Build and Run Real-Time Application” on page 1-15

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: '192.168.7.5'
        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`, `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: STOPPED
     DataAvailable: 0
```

`Stimulation` — Target computer stimulation control
stimulation control

The `Stimulation` property provides access to the `Target.Stimulation` object. To view the `Stimulation`, in the MATLAB Command Window, type:

```
tg.Stimulation
```

```
ans =

  StimulationControl with no properties.
```

`TargetStatus` — Target computer status
`TargetStatus` struct

The `TargetStatus` property provides access to target computer status information. The status values are enums. To view the `TargetStatus`, in the MATLAB Command Window, type:

```
tg.TargetStatus
```

```
ans =

  struct with fields:

    State: BUSY
    Error: ''
```

`ModelStatus` — Target computer model status
xxx

The `ModelStatus` property provides access to information about the loaded real-time application and related model. The status values are enums. To view the `ModelStatus`, in the MATLAB Command Window, type:

```
tg.ModelStatus

ans =

  struct with fields:

                    State: LOADED
              Application: 'slrt_ex_osc_outport'
                ModelName: 'slrt_ex_osc_outport'
                    Error: ''
                 LogLevel: "info"
          PollingThreshold: 1.0000e-04
            FileLogMaxRuns: 1
    OverrideBaseRatePeriod: 0
                 StopTime: 10
                 ExecTime: 0
                  TETInfo: [1×1 struct]
```

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 connecting a target computer by using the **Real-Time** tab in the Simulink Editor, Simulink Real-Time Explorer, or the `connect` function.
- `Disconnecting`, `Disconnected` – Events related to disconnecting a target computer by using the **Real-Time** tab in the Simulink Editor, Simulink Real-Time Explorer, or the `disconnect` function.
- `Installing`, `InstallFailed`, `Installed` – Events related to installing a real-time application on a target computer by using the `install` function.
- `Loading`, `LoadFailed`, `Loaded` – Events related to loading a real-time application 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 starting a real-time application 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 stopping a real-time application 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 rebooting a target computer by using the Simulink Real-Time Explorer or the `reboot` function.
- `UpdateBegin`, `UpdateMessage`, `UpdateFailed`, `UpdateCompleted` – Events related to updating target computer RTOS software by using the Simulink Real-Time Explorer or the `update` function.
- `SetIPAddressBegin`, `SetIPAddressFailed`, `SetIPAddressCompleted` – Events related to changing a target computer IP address by using the Simulink Real-Time Explorer or the `setipaddr` function.
- `StartupAppChanged` – Event related to changing a target computer startup application by using the Simulink Real-Time Explorer or the `setStartupApp` or `clearStartupApp` functions.
- `StopTimeChanged` – Event related to changing a real-time application stop time 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
copyPage	Copy one calibration page to another in the real-time application
deleteParamSet	Delete selected parameter set file from an application
deleteProfilerData	Delete execution profiler data from target computer
disconnect	Disconnect MATLAB from target computer
exportParamSet	Write ParameterSet object data to parameter set file
getAllInstruments	Get information on instruments added to target object
getApplicationFile	Get name of real-time application file
getAvailableProfile	Get information about available execution profiler data
getECUPage	Get current page number used by ECU on real-time application
getInstalledApplications	Get list of installed real-time application files
getLastApplication	Get name of real-time application most recently run on target computer
getNumPages	Get number of pages in memory for real-time application
getPersistentVariables	Get persistent variables from the Simulink Real-Time target computer to MATLAB
getProfilerData	Retrieve profile data object
getStartupApp	Get information about startup application configuration on target computer
getXCPPage	Get current page number used by XCP on real-time application
getparam	Read value of observable parameter in real-time application
getsignal	Read a signal value from a real-time application
getVersion	Get MATLAB, support package, and Speedgoat I/O Blockset version information
importParamSet	Create ParameterSet object
install	Install real-time application on target computer
isConnected	Get target computer connected status
isLoaded	Get real-time application loaded status
isRunning	Get real-time application running status
listParamSet	List available parameter set files for application
load	Deploy to target and load real-time application to target computer
loadParamSet	Restore parameter values saved in specified file
reboot	Restart target computer
removeAllInstruments	Remove instrument objects from target object
removeAllApplications	Removes all Simulink Real-Time applications from target computer
removeApplication	Removes Simulink Real-Time application from target computer
removeInstrument	Remove selected instrument object from target object
reset	Reset target object
resetProfiler	Reset profiling service state to Ready
saveParamSet	Save real-time application parameter values
setECUAndXCPPage	Set memory pages used by XCP and ECU to selected memory page on real-time application
setECUPage	Set memory page used by ECU to selected memory page on real-time application
setStartupApp	Configure startup real-time application for target computer
setStopTime	Configure stop time for real-time application
setXCPPage	Set memory page used by XCP to selected memory page on real-time application
setipaddr	Set IP address and netmask on the target computer

setparam	Change value of tunable parameter in real-time application
setPersistentVariables	Set persistent variables from MATLAB to the Simulink Real-Time target computer
start	Start execution of real-time application on target computer
startProfiler	Start profiling service on target computer
startRecording	Starts signal data live streaming and File Log logging
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
stopRecording	Stops signal data live streaming and File Log logging
update	Update RTOS version on target computer

Examples

Create Target Object for Default Target Computer

Create a target object that represents the default target computer.

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

```
tg = slrealtime
```

Create Target Object for Named Target Computer

Create a target object that represents target computer `TargetPC1`.

Create target object `tg` for a target computer by using an explicit name.

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

Version History

Introduced in R2020b

R2022b: Removed and Added Target Object Functions

Removed the `isRecording` utility function from the `Target` object.

These `Target` object utility functions have been added: `deleteParamSet` and `removeAllApplications`.

R2021b: Additional Target Object Functions

These `Target` object utility functions have been added: `deleteParamSet`, `getAllInstruments`, `getApplicationFile`, `getInstalledApplications`, `getLastApplication`, `isConnected`, `isLoaded`, `isRunning`, and `removeAllApplications`, `reset`.

These `Target` object ECU and XCP memory page functions have been added: `copyPage`, `getECUPage`, `getNumPages`, `getXCPPage`, `setECUAndXCPPage`, `setECUPage`, and `setXCPPage`. To support these functions, the default storage class for new models has changed from `default` to `PageSwitching` for model parameters and external parameters.

See Also

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

Topics

“Parameter Tuning and Data Logging”
“Create Listeners for Target Computer Events”
“Blocks Whose Outputs Depend on Inherited Sample Time”
“Target and Application Objects”

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.

When the `addInstrument()` function binds the instrument to a real-time application, Simulink Real-Time validates the signal names in the instrument. To reduce the time it takes to validate an instrument, add the instrument once to the target computer and run the real-time application as many times as needed. The instrument remains added to the target computer even after the real-time application stops and is reloaded.

Alternatively, you can add the instrument each time the real-time application runs. If using this approach, remove the instrument when the real-time application closes. For example, if adding code to an App Designer app, in the `CloseRequestFcn`, you could add a `removeInstrument(hInst)` function.

`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');
slbuild('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);
```

Start and Stop Streaming a Signal to SDI

To programmatically start and stop streaming a signal to the Simulation Data Inspector, you can use the `addInstrument` function and `removeInstrument` function.

```
% Create target object and build the real-time application.
tg = slrealtime;
connect(tg);
slbuild('slrt_ex_tank');

% Create instrument object and add a signal to it.
myInst = slrealtime.Instrument('slrt_ex_tank');
addSignal(myInst,'slrt_ex_tank/Controller',1);

% Load the real-time application and start.
load(tg,'slrt_ex_tank');
start(tg);

% To start streaming to the Simulation Data Inspector
% add the instrument to the target object.
addInstrument(tg,myInst);

% To stop streaming the signal, you can use removeInstrument.
removeInstrument(tg,myInst);
```

Input Arguments

`target_object` — Object that represents 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`

Version History

Introduced in R2020b

See Also

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

Topics

“Add App Designer App to Inverted Pendulum Model”

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');
connect(tg);
clearStartupApp(tg);
reboot(tg);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`Target` | `getStartupApp` | `setStartupApp`

Topics

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

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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Real-Time Application and Target Computer Modes”

copyPage

Package: `slrealtime`

Copy one calibration page to another in the real-time application

Syntax

```
copyPage(target_object,page_src,page_dst)
```

Description

`copyPage(target_object,page_src,page_dst)` copies the source calibration page to the destination calibration page in the real-time application that is loaded on the target computer.

Examples

Copy Calibration Page

Copy source calibration page 1 to destination calibration page 0.

```
copyPage(tg, 1, 0)
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`page_src` — Source calibration page

uint8

Source calibration page for copy operation.

`page_dst` — Destination calibration page

uint8

Destination calibration page for copy operation.

Version History

Introduced in R2021b

See Also

`Target` | `getECUPage` | `getNumPages` | `getXCPPage` | `setECUAndXCPPage` | `setECUPage` | `setXCPPage`

deleteParamSet

Package: `slrealtime`

Delete selected parameter set file from an application

Syntax

```
deleteParamSet(target_object,filename,app_name)
```

Description

`deleteParamSet(target_object,filename,app_name)` deletes a selected parameter set file on the target computer from the real-time application.

Examples

Delete Selected Parameter Set File

The `deleteParamSet` function deletes a parameter set file from the real-time application.

```
deleteParamSet(tg,'outportTypes','slrt_ex_osc_outport');
```

Input Arguments

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

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`filename` — Name of a parameter set file on the target computer
character vector | string scalar

Enter the name of the parameter set file from the target computer file system.

Example: `'outportTypes'`

Data Types: `char` | `string`

`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'`

Version History

Introduced in R2022b

See Also

`loadParamSet` | `saveParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

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 represents 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'`

Version History

Introduced in R2020b

See Also

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

Topics

“Execution Profiling for Real-Time Applications”

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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Real-Time Application and Target Computer Modes”

exportParamSet

Package: `slrealtime`

Write `ParameterSet` object data to parameter set file

Syntax

```
exportParamSet(target_object,parameter_set,app_name)
```

Description

`exportParamSet(target_object,parameter_set,app_name)` writes the parameter information from the `ParameterSet` object to the corresponding parameter file on the target computer. If the `app_name` is omitted, the currently loaded real-time application is used.

Examples

Export Parameters to Target Computer Parameter Set File

After tuning the parameters, export the modified parameter set to the target computer and load the parameters into the real-time application.

```
exportParamSet(tg,myParamSet);
loadParamSet(tg,myParamSet.filename);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`parameter_set` — `ParameterSet` object

`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

`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'`

Version History

Introduced in R2021a

See Also

`importParamSet` | `getparam` | `listParamSet` | `loadParamSet` | `saveParamSet` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

getAllInstruments

Package: `slrealtime`

Get information on instruments added to target object

Syntax

```
getAllInstruments(target_object)
```

Description

`getAllInstruments(target_object)` returns an Instrument array that has properties `AxesTimeSpan`, `AxesTimeSpanOverrun`, `Application`, and `ModelName`. The array provides information on all instruments that you have added to the target object.

Examples

Get All Instruments for Target Object

Get all instruments for target object `tg`.

```
getAllInstruments(tg)
```

```
ans =

  3×1 Instrument array with properties:

    AxesTimeSpan
    AxesTimeSpanOverrun
    Application
    ModelName
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target`

getApplicationFile

Package: `slrealtime`

Get name of real-time application file

Syntax

```
filename = getApplicationFile(target_object,app_name)
```

Description

`filename = getApplicationFile(target_object,app_name)` returns the development computer full path, file name, and extension of the MLDATX file for the real-time application that is installed on the target computer identified by the `target_object`.

If the MLDATX file for `app_name` is not found on the MATLAB path of the development computer, it is copied from the target computer to a temporary folder on the development computer.

If the MLDATX file for the real-time application is not installed on the target computer, the returned `filename` is empty.

Examples

Get File Name for Real-Time Application

Get the file name on the development computer of the real-time application that is installed on the target computer. This example is for a Windows® development computer.

```
filename = getApplicationFile(tg, "slrt_ex_osc")
```

```
filename =

    'C:\work\R2021b\myApps\slrt_ex_osc.mldatx'
```

Input Arguments

`target_object` — Object that represents 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

`filename` — Name of installed real-time application file in development computer file system
character vector

Returns the development computer full path, file name, and extension of the real-time application MLDATX file for the application installed on the target computer.

Version History

Introduced in R2021b

See Also

`Target`

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 represents 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..

Version History

Introduced in R2020b

See Also

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

Topics
“Execution Profiling for Real-Time Applications”

getECUPage

Package: `slrealtime`

Get current page number used by ECU on real-time application

Syntax

```
getECUPage(target_object)
```

Description

`getECUPage(target_object)` returns the current page number used by the embedded control unit (ECU) real-time application that is loaded on the target computer. This function returns the logical number for the calibration data page that is currently activated for the ECU.

Examples

Get ECU Calibration Page

Get the calibration page of the ECU.

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

```
ans =

  uint8

   0
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getNumPages` | `getXCPPage` | `setECUAndXCPPage` | `setECUPage` | `setXCPPage`

getNumPages

Package: `slrealtime`

Get number of pages in memory for real-time application

Syntax

```
getNumPages(target_object)
```

Description

`getNumPages(target_object)` returns the number of pages in memory for a real-time application that is loaded on the target computer.

Examples

Get Number of Pages in Memory

Get the number of pages in memory for the real-time application.

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

```
ans =

     2
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getECUPage` | `getXCPPage` | `setECUAndXCPPage` | `setECUPage` | `setXCPPage`

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');
model = 'slrt_ex_osc';
xfername = [model,'/Signal Generator'];
slbuild(model);
load(tg,model);
getparam(tg,xfername,'Amplitude')
```

```
ans =

     1
```

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');
model = 'slrt_ex_osc';
open_system(model);
```

```
Freq = Simulink.Parameter;
Freq.StorageClass = 'ExportedGlobal';
Freq.Value = 10;
xfername = [model,'/Signal Generator'];
set_param(xfername,'Frequency','Freq');
slbuild(model);
load(tg,model);
getparam(tg,'','Freq')

ans =

     10
```

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'`.

```
tg = slrealtime('TargetPC1');
model = 'slrt_ex_osc_struct';
open_system(model);
load('slrt_ex_osc_struct.mat');
slbuild(model);
load(tg,model);
getparam(tg,'','spkp')

ans =

  struct with fields:

    sg_freq: 20
    g2_gain: 1000000
    g1_gain: 400
     g_gain: 1000000
```

Get Global Parameter by Parameter Structure Field Name

Get the value of the MATLAB variable `'spkp.g_gain'` from the real-time application.

```
tg = slrealtime('TargetPC1');
model = 'slrt_ex_osc_struct';
open_system(model);
load('slrt_ex_osc_struct.mat');
slbuild(model);
load(tg,model);
getparam(tg,'','spkp.g2_gain')

ans =

    1000000
```

Access Parameter Values in Parameter Structure

The `getparam` and `setparam` functions support dot notation syntax to access parameter values in real-time applications. These are examples of more advanced syntax.

```
% If a parameter is a struct, a single element of any
% array can be specified at any arbitrary depth in the struct.
tg.setparam('', 'p.a.b(2).c', val)
val = tg.getparam('', 'p.a.b(2).c')

% If a parameter is an array of structs, one element of
% the struct array can be specified as follows:
tg.setparam('', 'p(2,2).x.y.z', val)
val = tg.getparam('', 'p(2,2).x.y.z')

% If a parameter is N dimensions, a single element of
% the parameter can be accessed by specifying each dimension.
tg.setparam('top/constant', 'Value(3,4)', val)
val = tg.getparam('top/constant', 'Value(3,4)')

% If a parameter is Mx1 or 1xN (row or column vector),
% the following syntax specifying a single index
% is allowed:
tg.setparam('top/constant1', 'Value(4)', val)
```

Input Arguments

`target_object` — Object that represents 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
number | character vector | string scalar | complex | structure | numeric array

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

Version History

Introduced in R2020b

R2021b: Parameter Structure for getparam and setparam

The operation of the `getparam` function and `setparam` function supports dot notation for:

- Specifying a field of a struct for `getparam`. It has the same support as `setparam`.
- Specifying an element of a struct array or matrix for `getparam` and `setparam`.
- Specifying one field of a struct when any substructure is an array of structs for `getparam` and `setparam`.

See Also

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

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

getPersistentVariables

Package: `slrealtime`

Get persistent variables from the Simulink Real-Time target computer to MATLAB

Syntax

```
variables_struct = getPersistentVariables(target_object)
```

Description

`variables_struct = getPersistentVariables(target_object)` gets persistent variables values from the persistent variables on the target computer and places these values into a structure of MATLAB variables on the development computer.

Examples

Get and Set Persistent Variables

The `getPersistentVariables` function and `setPersistentVariables` function enable you to access persistent variables that are created and updated in the real-time application by using the Persistent Variable Read block and Persistent Variable Write block.

1 Get persistent variable values from target computer `tg`.

```
myPersist = getPersistentVariables(tg)
```

```
myPersist =

     []
```

2 Change the value of `Variable1` in the `myPersist` structure. Add `Variable1` to the structure.

```
myPersist.Variable1 = 10;
myPersist.Variable2 = int8([1, 2; 3, 4]);
myPersist
```

```
myPersist =

  struct with fields:

    Variable1: 10
    Variable2: [2×2 int8]
```

3 Set the persistent variable values on the target computer. Get the variables from the target computer.

```
setPersistentVariables(tg,myPersist)
myMorePersist = getPersistentVariables(tg)
```

```
myMorePersist =

  struct with fields:
```

```
Variable1: 10
Variable2: [2×2 int8]
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`variables_struct` — Structure of persistent variable values

struct

Stores persistent variables from the target computer in a structure on the development computer.

Example: `myPersist`

Version History

Introduced in R2022a

See Also

`setPersistentVariables` | Persistent Variable Read | Persistent Variable Write

Topics

“Apply Persistent Variables in Real-Time Applications”

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 Profiler and the SLRT Overload Options block use different mechanisms to measure TET and do not generate identical TET values.

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.

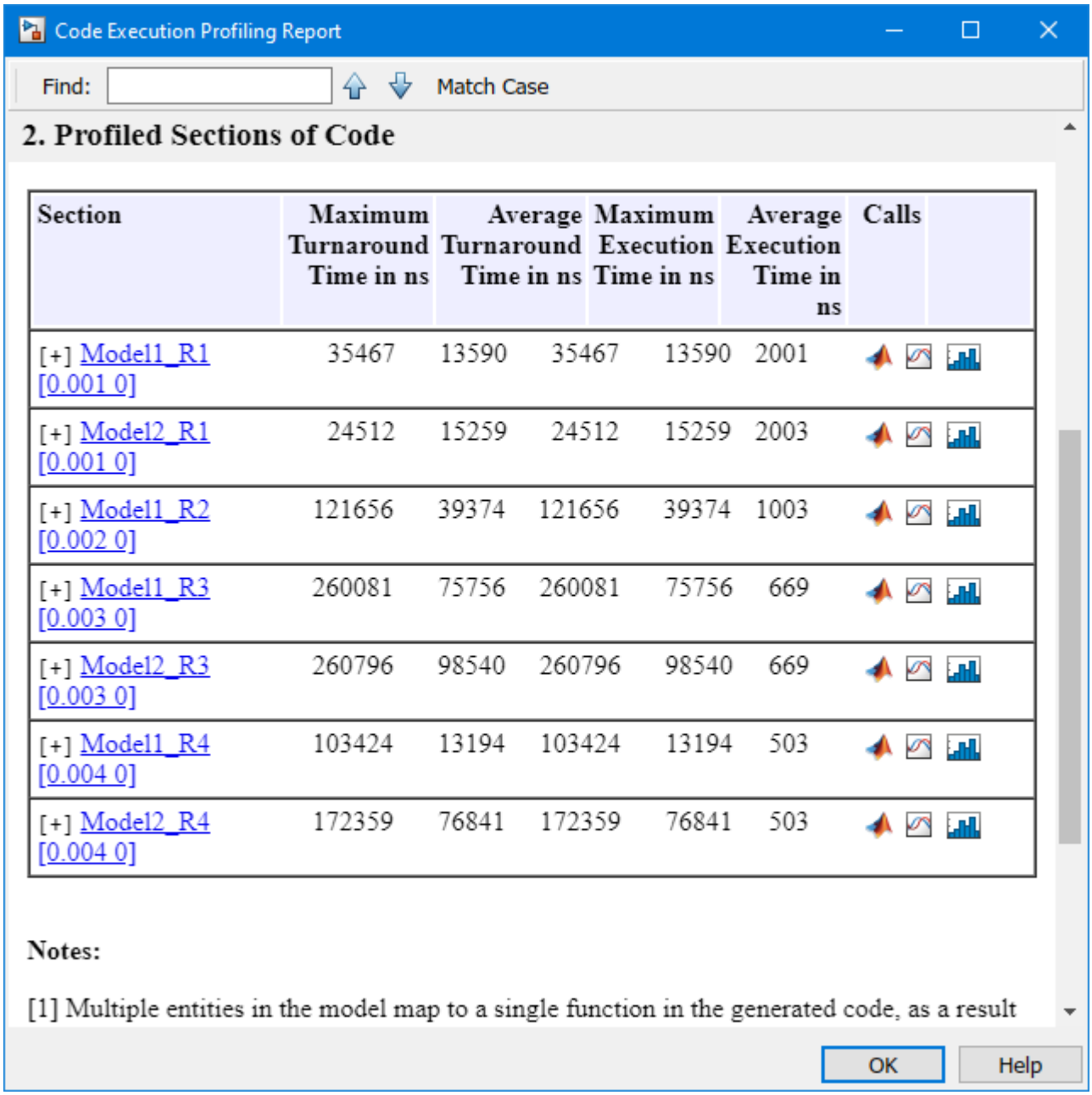
Examples

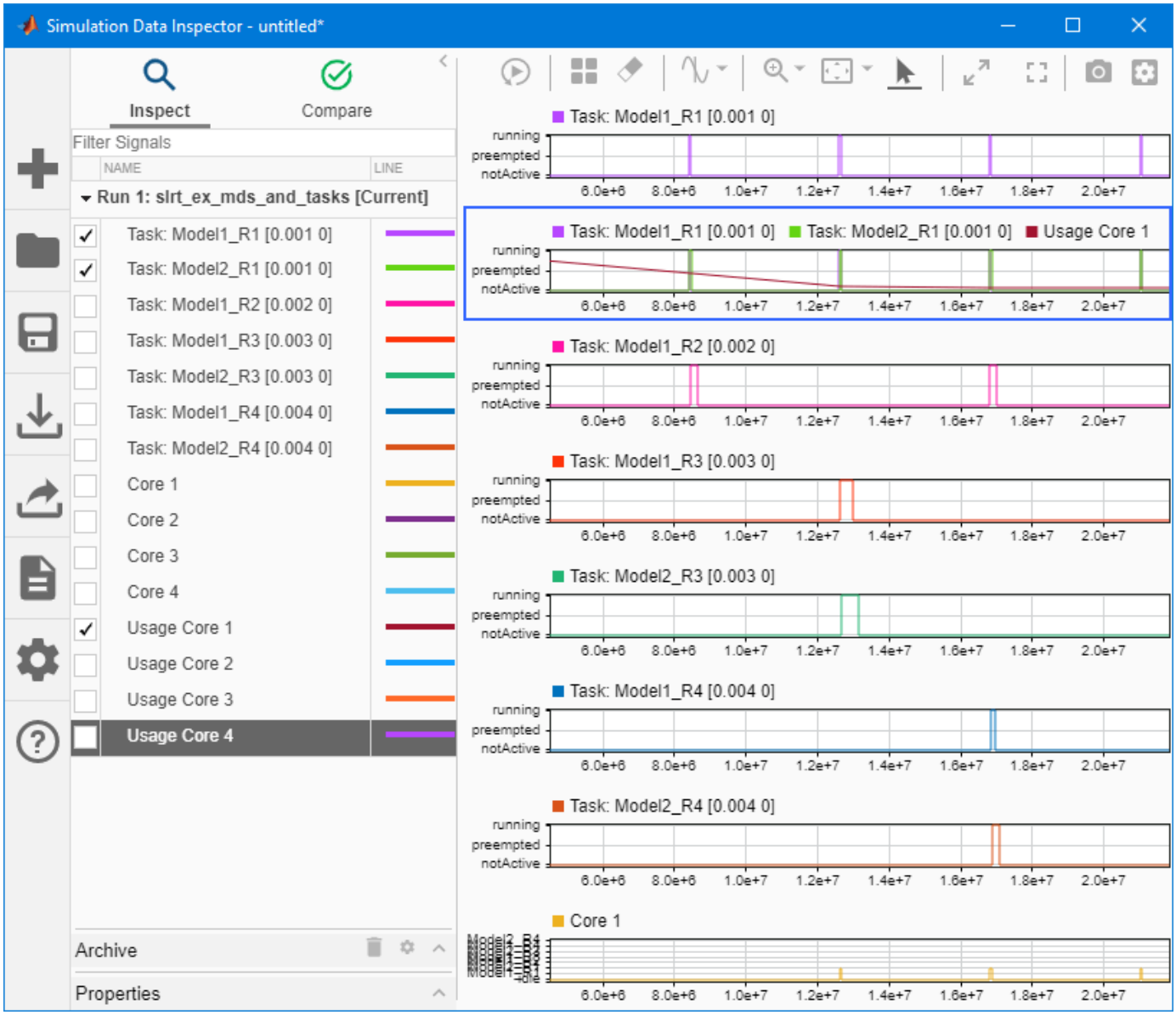
Run Profiler and Implicitly Display Profiler Data

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.

```
1   tg = slrealtime('TargetPC1');
    slbuild('slrt_ex_mds_and_tasks');
    load(tg,'slrt_ex_mds_and_tasks');
    startProfiler(tg);
    start(tg);
2   stopProfiler(tg);
    stop(tg);
```

```
3  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.
```





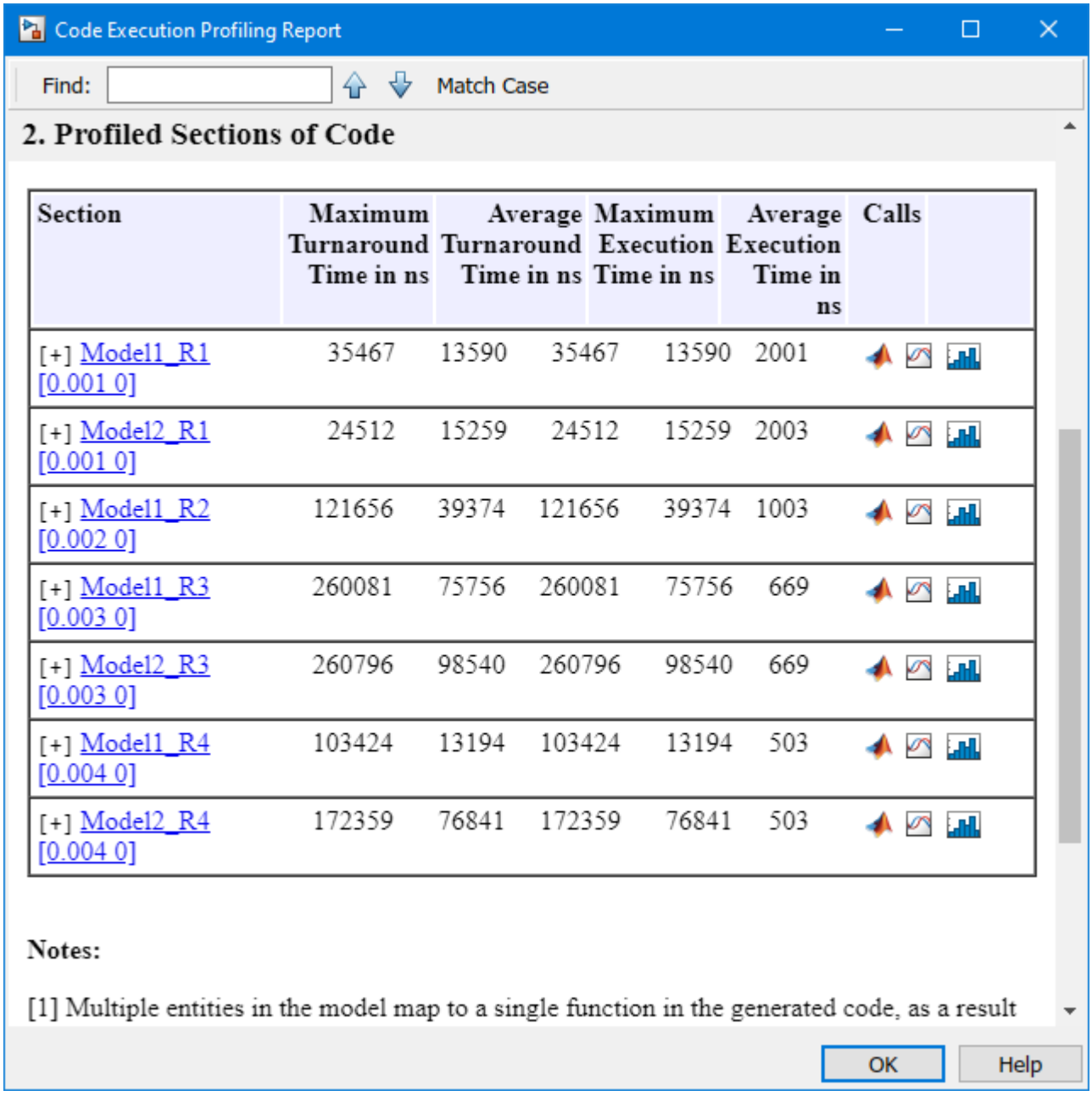
Run Profiler and Explicitly Display Profiler Data

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.

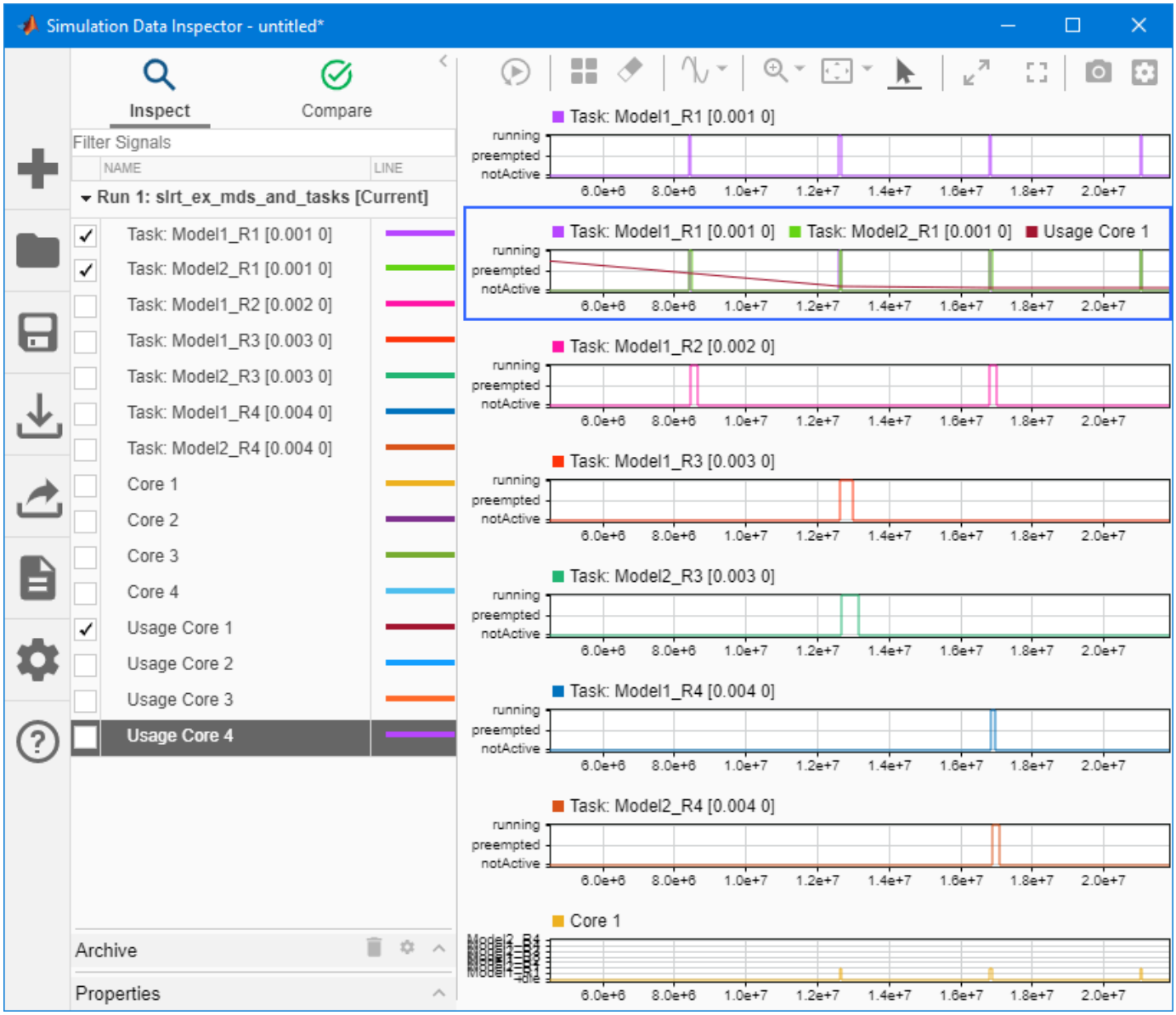
```
tg = slrealtime('TargetPC1');
slbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);
start(tg);
stopProfiler(tg);
stop(tg);
profiler_object = getProfilerData(tg);
```

```
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.
```

```
4 report(profiler_object);
```




```
5 plot(profiler_object);
```



Input Arguments

`target_object` — Object that represents 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.

Version History

Introduced in R2020b

See Also

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

Topics

“Execution Profiling for Real-Time Applications”

getsignal

Package: `slrealtime`

Read a signal value from a real-time application

Syntax

```
value = getsignal(target_object, blockPath, portIndex)
```

Description

`value = getsignal(target_object, blockPath, portIndex)` returns the value of the signal selected by the *`portIndex`* in block *`block_path`* from the real-time application that is loaded on the target computer. This function also supports multi-instance referenced models.

Examples

Get Signal Value by Using Port Index and Block Names

This example builds a real-time application from model `slrt_ex_osc`, loads the application on the target computer, and gets the value of the signal from block `'Signal Generator'` port 1.

```
tg = slrealtime('TargetPC1');
slbuild('slrt_ex_osc');
load(tg,'slrt_ex_osc');
getsignal(tg,'slrt_ex_osc/Signal Generator',1)
```

```
ans =

     0
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`blockPath` — Hierarchical name of the originating block

character vector | string

The *`block_path`* values can a character vector or string.

Example: `'slrt_ex_osc/Signal Generator'`

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

integer

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

Example: 1

Output Arguments

`value` — Value of signal

scalar | complex | structure

The value is the value of the signal in the real-time application. If the signal is a bus, a struct is returned. Correct data type, complexity, and dimensions are returned.

Version History

Introduced in R2021a

See Also

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

Topics

“Display and Filter Hierarchical Signals and Parameters”
“Troubleshoot Signals Not Accessible by Name”

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_mds_and_tasks')
setStartupApp(tg,'slrt_ex_mds_and_tasks')
getStartupApp(tg)
```

```
ans =

    'slrt_ex_mds_and_tasks'
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`Target` | `setStartupApp` | `clearStartupApp`

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

getVersion

Package: `slrealtime`

Get MATLAB, support package, and Speedgoat I/O Blockset version information

Syntax

```
version_info = getVersion(target_object)
```

Description

`version_info = getVersion(target_object)` returns the MATLAB version and release, support package version, and Speedgoat I/O Blockset version present on the development computer and target computer. The version information is returned as a structure.

Examples

Get Target Version Information

Get the MATLAB version and release, support package version, and Speedgoat I/O Blockset version present on the development computer and target computer.

```
version_info = getVersion(tg)
```

```
version_info =

  struct with fields:

       MatlabVersion: "R2023a"
    MatlabDescription: "Prerelease"
              Release: "9.14.0.xxxx"
    SupportPkgVersion: '22.1.0'
     SpeedgoatRelease: [Speedgoat I/O Blockset v9.4.0.3 build 26200]
           LibraryVer: []
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`version_info` — Version information

structure

Version information, specified as a structure. The structure provides information on the MATLAB version, support package version, and Speedgoat I/O Blockset version present on the development computer and target computer.

Version History

Introduced in R2023a

See Also

`Target`

Topics

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

getXCPPage

Package: `slrealtime`

Get current page number used by XCP on real-time application

Syntax

```
getXCPPage(target_object)
```

Description

`getXCPPage(target_object)` returns the current page number used by the Universal Measurement and Calibration Protocol (XCP) on a real-time application that is loaded on the target computer. This function returns the logical number for the calibration data page that is currently activated for XCP.

Examples

Get XCP Calibration Page

Get the calibration page for XCP.

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

```
ans =

  uint8

   0
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getECUPage` | `getNumPages` | `setECUAndXCPPage` | `setECUPage` | `setXCPPage`

importParamSet

Package: `slrealtime`

Create `ParameterSet` object

Syntax

```
parameter_set = importParamSet(target_object,filename,app_name)
```

Description

`parameter_set = importParamSet(target_object,filename,app_name)` imports the parameters from the parameter set file on the target computer into a `ParameterSet` object on the development computer. If the `app_name` is omitted, the currently loaded real-time application is used. When a real-time application stops, its parameter values are saved to parameter set file `autoSaveOnStop`. You can import this parameter set to the development computer and load it to the real-time application.

Examples

Import Parameters to Development Computer `ParameterSet` Object

Import the parameters from the parameter set file on the target computer into a `ParameterSet` object on the development computer.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
connect(tg);
load(tg,mdlName);
paramSetName = 'myParamSet';
saveParamSet(tg,paramSetName);
myParamSet = importParamSet(tg,paramSetName);
```

Import Parameters to Development Computer `ParameterSet` Object from Application Run

Import the parameters from the `autoSaveOnStop` parameter set file on the target computer into a `ParameterSet` object on the development computer.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
connect(tg);
load(tg,mdlName);
paramSetName = 'myParamSet';
saveParamSet(tg,paramSetName);
start(tg);
pause(3);
```

```
% the pause provides for the app to run and stop
myParamSet = importParamSet(tg,'autoSaveOnStop','slrt_ex_osc_outport');
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`filename` — Name of a parameter set file on the target computer

character vector | string scalar

Enter the name of the parameter set file from the target computer file system.

Example: `'outportTypes'`

Data Types: `char` | `string`

`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

`parameter_set` — `ParameterSet` object

`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

Version History

Introduced in R2021a

See Also

`exportParamSet` | `getparam` | `listParamSet` | `loadParamSet` | `saveParamSet` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

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.

You also can install 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”.

`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');
slbuild('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');
slbuild('slrt_ex_osc');
install(tg,'slrt_ex_osc','force');
```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2020b

See Also

`Target` | `start` | `stop`

Topics

“Real-Time Application and Target Computer Modes”

getInstalledApplications

Package: `slrealtime`

Get list of installed real-time application files

Syntax

```
app_names = getInstalledApplications(target_object)
```

Description

`app_names = getInstalledApplications(target_object)` returns the names of real-time applications installed on the target computer.

Examples

Get Names of Real-Time Applications

Get a list of the installed real-time applications on the target computer.

```
apps = getInstalledApplications(tg)
```

```
apps =

  1×1 cell array

    {'slrt_ex_osc'}
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Output Arguments

`app_names` — Real-time application names

array of character vectors

Provides names of real-time application MLDATX files that are installed on the target computer.

Version History

Introduced in R2021b

See Also

`Target`

getLastApplication

Package: `slrealtime`

Get name of real-time application most recently run on target computer

Syntax

```
getLastApplication(target_object)
```

Description

`getLastApplication(target_object)` returns the name of the real-time application that was most recently run on the target computer.

Examples

Get Last Application Run

Get the name of the real-time application that was most recently run on the target computer.

```
getLastApplication(tg)
```

```
ans =

    'slrt_ex_osc'
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target`

isConnected

Package: `slrealtime`

Get target computer connected status

Syntax

```
isConnected(target_object)
```

Description

`isConnected(target_object)` returns true if target object is connected to the target computer or returns false if target object is disconnected from the target computer.

Examples

Get Target Computer Connection Status

Get the target computer connection status.

```
isConnected(tg)
```

```
ans =

  logical

   1
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target`

isLoaded

Package: `slrealtime`

Get real-time application loaded status

Syntax

```
isLoaded(target_object,app_name)
```

Description

`isLoaded(target_object,app_name)` returns the real-time application loaded status of the target computer.

Examples

Get Real-Time Application Loaded Status

Get the loaded status of a real-time application on the target computer.

1. To get the loaded status of any real-time application on the target computer, use:

```
[LOADED, LOADEDAPPNAME] = isLoaded(tg)
```

```
LOADED =

  logical

   1


LOADEDAPPNAME =

    'slrt_ex_osc'
```

2. To get the loaded status of a real-time application that you select on the target computer, use:

```
[LOADED, LOADEDAPPNAME] = isLoaded(tg, 'slrt_ex_osc')
```

```
LOADED =

  logical

   1


LOADEDAPPNAME =

    'slrt_ex_osc'
```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2021b

See Also

`Target`

isRunning

Package: `slrealtime`

Get real-time application running status

Syntax

```
isRunning(target_object,app_name)
```

Description

`isRunning(target_object,app_name)` returns the running status of a real-time application on the target computer.

Examples

Get Real-Time Application Running Status

Get the real-time application running status and the name of the application.

1 To get the running status of any real-time application on the target computer, use:

```
[running, runningAppName] = isRunning(tg)
```

```
running =

  logical

   1


runningAppName =

    'slrt_ex_osc'
```

2 To get the running status of a real-time application that you select on the target computer, use:

```
[running, runningAppName] = isRunning(tg,'slrt_ex_osc')
```

```
running =

  logical

   1


runningAppName =

    'slrt_ex_osc'
```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2021b

See Also

`Target`

listParamSet

Package: `slrealtime`

List available parameter set files for application

Syntax

```
parameter_sets = listParamSet(target_object,app_name)
```

Description

`parameter_sets = listParamSet(target_object,app_name)` lists the parameter set files on the target computer for the real-time application.

Examples

List Available Parameter Set Files

The `listParamSet` function returns a list of parameter set files that are available for the real-time application.

```
myParamList = listParamSet(tg,'slrt_ex_osc_outport')
```

Input Arguments

`target_object` — Object that represents 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

`parameter_sets` — List of parameter set files
cell array of character vectors

The `listParamSet` function returns a list of parameter set files that are available for the real-time application.

Version History

Introduced in R2021a

See Also

`deleteParamSet` | `loadParamSet` | `saveParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

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.

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

   ```
   tg = slrealtime('TargetPC1');
   ```
2. Build the real-time application.

   ```
   slbuild('slrt_ex_osc');
   ```
3. Load the real-time application.

   ```
   load(tg,'slrt_ex_osc');
   ```
4. If the real-time application MLDATX file is not located in your current folder, add path information to the `load` function. For example,

   ```
   appPath = fileparts(which('slrt_ex_osc.mldatx'));
   load(tg, [appPath '\' 'slrt_ex_osc']);
   ```
5. Start the application.

   ```
   start(tg);
   ```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2020b

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”

loadParamSet

Package: `slrealtime`

Restore parameter values saved in specified file

Syntax

```
loadParamSet(target_object,filename,page)
loadParamSet(target_object,parameter_set.filename,page)
```

Description

`loadParamSet(target_object,filename,page)` loads the parameter values into the given memory page of the loaded real-time application from a parameter set file on the target computer.

You also can load a parameter set into a 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”.

`loadParamSet(target_object,parameter_set.filename,page)` loads the parameter values into the given memory page of the loaded real-time application from a parameter set file that is identified by the parameter set `filename` property.

Examples

Load Saved Parameters into Application

Load parameters from the parameter set file into the loaded real-time application.

```
% load real-time application
mdlName = 'slrt_ex_osc_outport';
tg = slrealtime('TargetPC1');
load(tg,mdlName);

% load a previously saved
% parameter set file
paramSetName = 'outportTypes';
loadParamSet(tg,paramSetName);
```

Get Parameters from Parameter Set Object and Load

Load parameters from the parameter set file into the loaded real-time application.

```
% load real-time application
mdlName = 'slrt_ex_osc_outport';
tg = slrealtime('TargetPC1');
load(tg,mdlName);
```

```
% get parameter values from previously created
% ParameterSet object and load
exportParamSet(tg,myParamSet);
loadParamSet(tg,myParamSet.filename);
```

Get Tuned Parameters from Previous Run and Load

Load tuned parameters from the previous run of the real-time application into the loaded application.

```
% load real-time application
% that has been run at least once
% and has the auto save on stop option enabled
mdlName = 'slrt_ex_osc_outport';
tg = slrealtime('TargetPC1');
load(tg,mdlName);

% load tuned parameter values from previous run
% of real-time application that were auto saved
% when the application was stopped
loadParamSet(tg,'autoSaveOnStop');
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`filename` — Name of a parameter set file on the target computer

character vector | string scalar

Enter the name of the parameter set file from the target computer file system.

Example: `'outportTypes'`

Data Types: `char` | `string`

`page` — memory page number

`0` (default) | int8

(Optional.) Select memory page of the loaded real-time application for parameters.

Example: `0`

`parameter_set` — `ParameterSet` object

`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

Version History

Introduced in R2021a

See Also

`deleteParamSet` | `listParamSet` | `saveParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”
“Save and Reload Parameters by Using Simulink Real-Time Explorer”

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.

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

   ```
   tg = slrealtime('TargetPC1');
   ```
2. Restart target computer.

   ```
   reboot(tg);
   ```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

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”

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');
slbuild('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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics
“Add App Designer App to Inverted Pendulum Model”

removeAllApplications

Package: `slrealtime`

Removes all Simulink Real-Time applications from target computer

Syntax

```
removeAllApplications(target_object)
```

Description

`removeAllApplications(target_object)` removes all the real-time applications and associated files from the target computer. If one of the real-time applications was configured as the startup application, the function clears the startup application selection on the target computer. To be removed, a real-time application cannot be loaded or running.

Examples

Remove All Real-Time Applications

The `removeAllApplications` function removes all installed applications from the target computer.

```
removeAllApplications(tg)
```

Input Arguments

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

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2022b

See Also

`Target` | `install` | `removeApplication`

removeApplication

Package: `slrealtime`

Removes Simulink Real-Time application from target computer

Syntax

```
removeApplication(target_object,app_name)
```

Description

`removeApplication(target_object,app_name)` removes the selected real-time application and associated files from the target computer. If the real-time application was configured as the startup application, the function clears the startup application selection on the target computer. To be removed, the real-time application cannot be loaded or running.

Examples

Remove Real-Time Application

The `removeApplication` function removes an installed application from the target computer.

```
removeApplication(tg,'slrt_ex_osc')
```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2022a

See Also

`Target` | `install` | `removeAllApplications`

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');
slbuild('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 represents 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`

Version History

Introduced in R2020b

See Also

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

Topics

“Add App Designer App to Inverted Pendulum Model”

reset

Package: `slrealtime`

Reset target object

Syntax

```
reset(target_object)
```

Description

`reset(target_object)` resets the `Target` object. This function disconnects the `Target` object from the target computer and resets the internal state of the `Target` object. Use this function to re-establish `Target` object operation when some issue stops communication between the `Target` object and target computer.

Examples

Reset Target Object

Reset the target object `tg`.

```
reset(tg);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2021b

See Also

`Target`

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'))
slbuild('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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`ProfilerData` | `Target` | Enable Profiler

Topics
“Execution Profiling for Real-Time Applications”

saveParamSet

Package: `slrealtime`

Save real-time application parameter values

Syntax

```
saveParamSet(target_object, filename,page)
```

Description

`saveParamSet(target_object, filename,page)` saves the parameter values from the given memory page of the loaded real-time application into a parameter set file on the target computer.

You also can save a parameter set from a 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

Save Parameters from Application to Parameter Set File

Save parameters from the loaded application `slrt_ex_osc_outport` to a file named `'myParamSet'`.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
connect(tg);
load(tg,mdlName);
paramSetName = 'myParamSet';
saveParamSet(tg,paramSetName,1);
myParamSet = importParamSet(tg,paramSetName);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`filename` — Name of a parameter set file on the target computer

character vector | string scalar

Enter the name of the parameter set file from the target computer file system.

Example: `'outportTypes'`

Data Types: `char` | `string`

page — memory page number
`0` (default) | int8

(Optional.) Select memory page of the loaded real-time application for parameters.

Example: `0`

Version History

Introduced in R2021a

See Also

`deleteParamSet` | `listParamSet` | `loadParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”
“Save and Reload Parameters by Using Simulink Real-Time Explorer”

setECUAndXCPPage

Package: `slrealtime`

Set memory pages used by XCP and ECU to selected memory page on real-time application

Syntax

```
setECUAndXCPPage(target_object,page_num)
```

Description

`setECUAndXCPPage(target_object,page_num)` sets the memory pages used by the Universal Measurement and Calibration Protocol (XCP) and the Engine Control Unit (ECU) to the selected memory page on a real-time application that is loaded on the target computer.

Examples

Set ECU and XCP Calibration Page

Set the ECU and XCP calibration page to page 0.

```
setECUAndXCPPage(tg, 0)
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`page_num` — Calibration page number

uint8

Selects the page number for the ECU and XCP calibration page.

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getECUPage` | `getNumPages` | `getXCPPage` | `setECUPage` | `setXCPPage`

setECUPage

Package: `slrealtime`

Set memory page used by ECU to selected memory page on real-time application

Syntax

```
setECUPage(target_object,page_num)
```

Description

`setECUPage(target_object,page_num)` sets the memory page used by the Engine Control Unit (ECU) to the selected memory page on a real-time application that is loaded on the target computer.

Examples

Set ECU Calibration Page

Set the ECU calibration page to page 0.

```
setECUPage(tg, 0)
```

Input Arguments

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

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`page_num` — Calibration page number
uint8

Selects the page number for the ECU calibration page.

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getECUPage` | `getNumPages` | `getXCPPage` | `setECUAndXCPPage` | `setXCPPage`

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 `'192.168.7.5'` and the netmask to `'255.255.255.0'`. These values are retained by the target computer.

```
tg = slrealtime('TargetPC1');
setipaddr(tg,'192.168.7.5','255.255.255.0');
reboot(tg);
```

Input Arguments

`target_object` — Object that represents 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: `'192.168.7.5'`

`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'`

Version History

Introduced in R2020b

See Also

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

Topics

“Real-Time Application and Target Computer Modes”

setparam

Package: `slrealtime`

Change value of tunable parameter in real-time application

Syntax

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

Description

`setparam(target_object, block_path, parameter_name, parameter_value,'Force',true)` 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,'Force',true)` 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');
model = 'slrt_ex_osc';
xfername = [model,'/Signal Generator'];
slbuild(model);
load(tg,model);
setparam(tg,xfername,'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');
model = 'slrt_ex_osc';
xfername = [model,'/Signal Generator'];
slbuild(model);
load(tg,model);
for i = 1 : 3
    setparam(tg,xfername,'Amplitude',(i*2))
end
```

Set Global Parameter by Scalar Parameter Name

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

```
tg = slrealtime('TargetPC1');
model = 'slrt_ex_osc';
open_system(model);
Freq = Simulink.Parameter;
Freq.StorageClass = 'ExportedGlobal';
Freq.Value = 10;
xfername = [model,'/Signal Generator'];
set_param(xfername,'Frequency','Freq');
slbuild(model);
load(tg,model);
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');
model = 'slrt_ex_osc_struct';
open_system(model);
load('slrt_ex_osc_struct.mat');
slbuild(model);
load(tg,model);
setparam(tg,'','spkp.g2_gain',10000000)
```

Access Parameter Values in Parameter Structure

The `getparam` and `setparam` functions support dot notation syntax to access parameter values in real-time applications. These are examples of more advanced syntax.

```
% If a parameter is a struct, a single element of any
% array can be specified at any arbitrary depth in the struct.
tg.setparam('', 'p.a.b(2).c', val)
val = tg.getparam('', 'p.a.b(2).c')

% If a parameter is an array of structs, one element of
% the struct array can be specified as follows:
tg.setparam('', 'p(2,2).x.y.z', val)
val = tg.getparam('', 'p(2,2).x.y.z')

% If a parameter is N dimensions, a single element of
% the parameter can be accessed by specifying each dimension.
tg.setparam('top/constant', 'Value(3,4)', val)
val = tg.getparam('top/constant', 'Value(3,4)')

% If a parameter is Mx1 or 1xN (row or column vector),
% the following syntax specifying a single index
```

```
% is allowed:
tg.setparam('top/constant1', 'Value(4)', val)
```

Input Arguments

`target_object` — Object that represents 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 | numeric array

New value with data type as required by parameter.

Example: `1`

`Force` — Force set value

`false` (default) | `true`

The optional `'Force', true` name-value pair argument forces the parameter value set operation even if the value is not in the range of `[min max]` for the parameter.

Example: `'Force', true`

Data Types: `logical`

Version History

Introduced in R2020b

R2021b: Parameter Structure for getparam and setparam

The operation of the `getparam` function and `setparam` function supports dot notation for:

- Specifying a field of a struct for `getparam`. It has the same support as `setparam`.
- Specifying an element of a struct array or matrix for `getparam` and `setparam`.
- Specifying one field of a struct when any substructure is an array of structs for `getparam` and `setparam`.

See Also

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

Topics

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

setPersistentVariables

Package: `slrealtime`

Set persistent variables from MATLAB to the Simulink Real-Time target computer

Syntax

```
setPersistentVariables(target_object,variables_struct)
```

Description

`setPersistentVariables(target_object,variables_struct)` sets persistent variables to values from MATLAB variables on the development computer into the persistent variables on the target computer. The variables can be empty or a struct whose fields are persistent variables.

Examples

Get and Set Persistent Variables

The `getPersistentVariables` function and `setPersistentVariables` function enables you to access persistent variables that are created and updated in the real-time application by using the Persistent Variable Read block and Persistent Variable Write block.

1 Get persistent variable values from target computer `tg`.

```
myPersist = getPersistentVariables(tg)
```

```
myPersist =

     []
```

2 Change the value of `Variable1` in the `myPersist` structure. Add `Variable1` to the structure.

```
myPersist.Variable1 = 10;
myPersist.Variable2 = int8([1, 2; 3, 4]);
myPersist
```

```
myPersist =

  struct with fields:

    Variable1: 10
    Variable2: [2×2 int8]
```

3 Set the persistent variable values on the target computer. Get the variables from the target computer.

```
setPersistentVariables(tg,myPersist)
myMorePersist = getPersistentVariables(tg)
```

```
myMorePersist =

  struct with fields:
```

```
Variable1: 10
Variable2: [2×2 int8]
```

Remove Persistent Variables from Target Computer

You can use the `setPersistentVariables` function to remove the persistent variables that are stored on the target computer.

1. On the development computer, create a `Target` object `tg` and connect to the target computer.

   ```
   tg = slrealtime;
   connect(tg);
   ```
2. Use the `setPersistentVariables` function to clear the persistent variable values that are stored on the target computer.

   ```
   setPersistentVariables(tg,[]);
   ```

Remove a Persistent Variable from Target Computer

You can use the `setPersistentVariables` function to remove a persistent variable that is stored on the target computer.

1. On the development computer, create a `Target` object `tg` and connect to the target computer.

   ```
   tg = slrealtime;
   connect(tg);
   ```
2. Get the persistent variable values from the target computer.

   ```
   myPersistVars = getPersistentVariables(tg);
   ```
3. Copy the persistent variable values to a variable and remove the field for the variable. This example removes the field for the variable `position`.

   ```
   myNewPersistVars = rmfield(myPersistVars,'position');
   ```
4. Use the `setPersistentVariables` function to apply the updated persistent variable values to the target computer.

   ```
   setPersistentVariables(tg,myNewPersistVars);
   ```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`variables_struct` — Structure of persistent variable values

struct

A MATLAB struct whose field names are persistent variable names to set and whose field values are persistent variable values to set.

Example: `myPersist`

Version History

Introduced in R2022a

See Also

`getPersistentVariables` | Persistent Variable Read | Persistent Variable Write

Topics

“Apply Persistent Variables in Real-Time Applications”

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 represents 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'`

Version History

Introduced in R2020b

See Also

`Target` | `getStartupApp` | `clearStartupApp`

Topics
“Real-Time Application and Target Computer Modes”

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 represents 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

Version History

Introduced in R2020b

See Also

`Target` | `start` | `stop`

Topics
“Real-Time Application and Target Computer Modes”

setXCPPage

Package: `slrealtime`

Set memory page used by XCP to selected memory page on real-time application

Syntax

```
setXCPPage(target_object,page_num)
```

Description

`setXCPPage(target_object,page_num)` sets the memory page used by the Universal Measurement and Calibration Protocol (XCP) to the selected memory page on a real-time application that is loaded on the target computer.

Examples

Set XCP Calibration Page

Set the XCP calibration page to page 0.

```
setXCPPage(tg, 0)
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`page_num` — Calibration page number

uint8

Selects the page number for the XCP calibration page.

Version History

Introduced in R2021b

See Also

`Target` | `copyPage` | `getECUPage` | `getNumPages` | `getXCPPage` | `setECUAndXCPPage` | `setECUPage`

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 represents target computer
`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`Target` | `Targets`

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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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: `start(tg,'LogLevel','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: `start(tg,'PollingThreshold',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.

Note You can inadvertently delete existing file logs for an installed real-time application on the target computer if you use the `slrealtime.Application` function to change the Options for `FileLogMaxRuns` and then reload the application. To change the number of stored logs without deleting existing logs, load the real-time application and then change the FileLogMaxRuns option by using the `start(tg)` function.

Example: `start(tg,'FileLogMaxRuns',1)`

`StartStimulation` — Stimulation of inports and Playback blocks
`'on'` (default) | `'off'`

Select whether stimulation of root inports and Playback blocks starts when you start the real-time application.

Example: `start(tg,'StartStimulation','on')`

`StopTime` — Real-time application stop time
`StopTime` config set value (default)

Select stop time value for the real-time application.

Example: `start(tg,'StopTime',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: `start(tg,'ReloadOnStop',false)`

`AutoImportFileLog` — Configure file log import
`true` (default) | `false`

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

If a model includes a Enable File Log block, the `start(tg)` `AutoImportFileLog` option has no effect, and the `startRecording` function and `stopRecording` function only control signal streaming (not File Log logging).

Example: `start(tg,'AutoImportFileLog',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: `start(tg,'ExportToBaseWorkspace',true)`

Version History

Introduced in R2020b

R2022b: Stimulation Start and Stop Control

The `start` function `StartStimulation` option lets you select whether the real-time application starts with stimulation on or off for inports and Playback blocks

See Also

`Target` | `stop` | `load`

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

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.

When the profiler starts and stops, there is an increase in task execution time (TET) that can cause a CPU overload condition. If using profiler causes CPU overload, you can increase the sample time in order to use the profiler.

Examples

Profile Execution of Real-Time Application

Build the real-time application `slrt_ex_mds_and_tasks`. Load the real-time application. Start the profiler. Start the application.

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

% start profiler before starting application

start(tg);
```

Check Profiler Status from Target Object Property

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

1
```
tg = slrealtime('TargetPC1');
slbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
tg.ProfilerStatus
```

```
ans =

    'Ready'
```

2 Start the profiler, and then start the application.

```
startProfiler(tg);

% start profiler before starting application

start(tg);
```

3 After the application stops, check the profiler status.

```
tg.ProfilerStatus
```

```
ans =

    'DataAvailable'
```

Input Arguments

`target_object` — Object that represents 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'`

Version History

Introduced in R2020b

See Also

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

Topics
“Execution Profiling for Real-Time Applications”

startRecording

Package: `slrealtime`

Starts signal data live streaming and File Log logging

Syntax

```
startRecording(target_object)
```

Description

`startRecording(target_object)` starts a simulation run in the Simulation Data Inspector, enables signal data live streaming, and enables signal data File Log logging. Every time that signal logging and streaming is enabled by using the `startRecording` function, **Start Recording** button, or enabling the Enable File Log block creates a log run on the target computer.

When a model includes an Enable File Log block, the `startRecording` function and `stopRecording` function only control signal streaming and do not control file logging. For such a model, the `AutoImportFileLog` option does not import file log data into the Simulation Data Inspector on a `stopRecording`.

When a boolean signal in the model triggers the Enable File Log block, new Logs are created on the target. Retention of these new logs depends on the `FileLogMaxRuns` setting.

For more information about the file logging workflow, see “Signal Logging and Streaming Basics”.

Examples

Start Recording on Target Computer

Use the `startRecording` function to start recording on the target computer. You also can use the **Start Recording** button on the **Real-Time** tab in the Simulink Editor or in the Simulink Real-Time Explorer.

```
tg = slrealtime;
load(tg,'slrt_ex_osc')
start(tg);
stopRecording(tg);
startRecording(tg);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2022a

R2022b: Added Dashboard Block Compatibility

Added compatibility with Simulink dashboard blocks to the `startRecording` function and `stopRecording` function. When a model includes Simulink dashboard blocks, using the `startRecording` function or `stopRecording` function does not generate an error.

R2022b: Made Start and Stop Logging and Streaming Independent of Real-Time Application

You can set the start or stop state for logging and streaming independent of the run or stop state of the real-time application. This independence lets you start a real-time application with logging and streaming stopped by using the `stopRecording` function or the **Stop Recording** button on the **Real-Time** tab in the Simulink Editor or in the Simulink Real-Time Explorer.

See Also

`Target` | File Log | Enable File Log | `import` | `stopRecording`

Topics

“Signal Logging and Streaming Basics”

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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Real-Time Application and Target Computer Modes”

stop

Package: `slrealtime`

Stop execution of real-time application on target computer

Syntax

```
stop(target_object,AutoImportFileLog)
```

Description

`stop(target_object,AutoImportFileLog)` 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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`AutoImportFileLog` — Configure file log import

`true` (default) | `false`

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

If a model includes a Enable File Log block, the `stop(tg)` `AutoImportFileLog` option has no effect, and the `startRecording` function and `stopRecording` function only control signal streaming (not File Log logging).

Example: `stop(tg,'AutoImportFileLog',true)`

Version History

Introduced in R2020b

R2023a: AutoImportFileLog Control

The `stop` function `AutoImportFileLog` option lets you select whether the Simulink Real-Time auto imports file log data when the real-time application stops. This option lets you change the auto import operation that was selected by the `start` function.

See Also

`Target` | `start` | `load`

Topics

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

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.

When the profiler starts and stops, there is an increase in task execution time (TET) that can cause a CPU overload condition. If using profiler causes CPU overload, you can increase the sample time in order to use the profiler.

Examples

Start and Stop Profiler

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

1
```
tg = slrealtime('TargetPC1');
slbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
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);
```
2 At this point, call either the `getProfilerData` function or the `resetProfiler` function.

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Execution Profiling for Real-Time Applications”

stopRecording

Package: `slrealtime`

Stops signal data live streaming and File Log logging

Syntax

```
stopRecording(target_object)
```

Description

`stopRecording(target_object)` disables logging in the application and closes the run in the Simulation Data Inspector. The function disables the File Log logging, imports (based on the AutoImportFileLog flag) the File Log data into the Simulation Data Inspector in the current run, exports the data to base workspace based on the flag `'ExportToBaseWorkspace'`, and stops live streaming. Every time that signal logging and streaming is disabled by using the `stopRecording` function, **Stop Recording** button, or disabling the Enable File Log block closes a log run on the target computer.

The `stopRecording` function operation responds to the setting of the `AutoImportFileLog` option. The File Log data from the target is imported if the `AutoImportFileLog` option is true when you use the `stopRecording` function.

If the model includes an Enable File Log block, the `stopRecording` function only stops signal streaming. The block controls signal logging. Also, if the model includes an Enable File Log block, the `AutoImportFileLog` option has no effect.

For more information about the file logging workflow, see “Signal Logging and Streaming Basics”.

Examples

Stop Recording on Target Computer

Use the `stopRecording` function to stop recording on the target computer. You also can use the **Stop Recording** button on the **Real-Time** tab in the Simulink Editor or in the Simulink Real-Time Explorer.

```
tg = slrealtime;
load(tg,'slrt_ex_osc')
start(tg);
stopRecording(tg);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2022a

R2022b: Added Dashboard Block Compatibility

Added compatibility with Simulink dashboard blocks to the `startRecording` function and `stopRecording` function. When a model includes Simulink dashboard blocks, using the `startRecording` function or `stopRecording` function does not generate an error.

R2022b: Made Start and Stop Logging and Streaming Independent of Real-Time Application

You can set the start or stop state for logging and streaming independent of the run or stop state of the real-time application. This independence lets you start a real-time application with logging and streaming stopped by using the `stopRecording` function or the **Stop Recording** button on the **Real-Time** tab in the Simulink Editor or in the Simulink Real-Time Explorer.

See Also

`Target` | File Log | Enable File Log | `import` | `startRecording`

Topics

“Signal Logging and Streaming Basics”

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. When you update the RTOS on the target computer, the process removes the target computer `applications` folder and the installed real-time application MLDATX files.

`update(target_object,'force',true)` forces an update of all RTOS files on the target computer to the current version. When you update the RTOS on the target computer, the process removes the target computer `applications` folder and the installed real-time application MLDATX files.

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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Real-Time Application and Target Computer Modes”

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.
- 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.

Note The `Importing` property will be removed in a future release.

Example: `0`

`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

discard	Delete file log data from target computer
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);
```

Version History

Introduced in R2020b

R2022b: Removed Importing property

The `Importing` property of the `Target.FileLog` object is removed. An error appears when you use this property.

R2021b: Synchronous Processing for File Log Import

The operation of the `Target.FileLog` object has changed. Now, the file log import process is synchronous in MATLAB, which means that while data import is occurring, the MATLAB status is busy. The abort function has been removed. The `ImportProgress` property has been removed.

See Also

`Target` | `discard` | `list` | `import` | `startRecording` | `stopRecording`

Topics

“Parameter Tuning and Data Logging”

“Signal Logging and Streaming Basics”

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.

1. 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);
   ```
2. 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.

1 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
```

2 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`.

Version History

Introduced in R2020b

See Also

File Log | Enable File Log | `Target` | `list` | `import`

Topics
“Signal Logging and Streaming Basics”

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.

1
```
my_run_info = list(tg.FileLog)
```

```
my_run_info =

  3×3 table

            Application             StartDate          Size (in MB)
          ___________________    ____________________    _____

    1.    "slrt_ex_osc_rt_t"     12-Dec-2019 21:59:31    9.4944
    2.    "slrt_ex_osc_rt_t"     12-Dec-2019 21:59:45    8.4736
    3.    "slrt_ex_osc"          12-Dec-2019 21:59:57    8.2176
```

2 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 represents 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`.

Version History

Introduced in R2020b

See Also

File Log | Enable File Log | `Target` | `import` | `discard`

Topics
“Signal Logging and Streaming Basics”

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. You can import file log data from the target computer when the real-time application is stopped or recording is stopped.

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 `startRecording`.

To control file logging with the Enable File Log block, when you load the real-time application on the target computer, disable or enable file logging by using the **E** (enable/disable) input signal to the block.

If you started the real-time application with the `AutoImportFileLog` option enabled, 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 by using the `import` function.

The `AutoImportFileLog` option has no effect when using the `startRecording` function or `stopRecording` function.

If a model includes an Enable File Log block, the `AutoImportFileLog` option has no effect. Use the `import` function to upload logging data for real-time applications that you build from these models.

For more information about the file logging workflow, see “Signal Logging and Streaming Basics”.

Note: The `start` function `FileLogMaxRuns` option determines how many runs of file log data are retained on the target computer. As you add a new log for a simulation run, the oldest log exceeding the maximum number of runs is removed. To avoid losing file log data, use the `import` function or `AutoImportFileLog` option to upload logging data. For more information, see `FileLogMaxRuns`.

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.

1. 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);
   ```

2. 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`.

1. 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
   ```

2. 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 represents 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`.

Version History

Introduced in R2020b

R2022b: Backward Compatibility and Log Retention

The `import` function imports a file log that was created by the same release of MATLAB as the function.

The `import` function does not delete the data from the target computer after importing the data.

See Also

File Log | Enable File Log | `Target` | `list` | `discard` | `startRecording` | `stopRecording`

Topics
“Signal Logging and Streaming Basics”

slrealtime.fileLogImport

Package: `slrealtime`

After copying file logs from target to development computer, import file logs into Simulation Data Inspector

Syntax

```
slrealtime.fileLogImport(app_name,'Directory',apps_path)
slrealtime.fileLogImport(app_name)
slrealtime.fileLogImport(run_table)
slrealtime.fileLogImport(run_number)
```

Description

`slrealtime.fileLogImport(app_name,'Directory',apps_path)` imports file logs into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the specified folder `apps_path`.

Use the `slrealtime.fileLogImport` workflows when working with a standalone target computer that does not connect to Simulink Real-Time. If working with a target computer that connects to Simulink Real-Time on the development computer, use the `import` function or **Import File Log** button in the Simulink Real-Time Explorer.

`slrealtime.fileLogImport(app_name)` imports the logs for the selected real-time application name (string) into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the current folder `pwd`.

`slrealtime.fileLogImport(run_table)` imports the file logs for the selected run table (table) into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the current folder `pwd`.

`slrealtime.fileLogImport(run_number)` imports the file logs for the select row number (numeric) into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the current folder `pwd`.

Examples

Build and Run Real-Time Application

1. Open model `slrt_ex_osc`.
2. In the Simulink Editor, from the **Real-Time** tab, click **Hardware Settings**.
3. In the **Simulink Real-Time Options** pane, change **Max file log runs** to 5 and click OK.
4. Click **Run on Target**.
5. After the run ends, close the model and exit MATLAB.

Create File Logs on Target Computer

Because this example shows how to use the `slrealtime.fileLogImport` function to import file logs that are created on a disconnected target computer, the example shows how to start the real-time example by using an SSH session from the target computer command line. If you are starting the real-time application from MATLAB and are using `slrealtime.fileLogImport`, consider using the `'FileLogMaxRuns'` argument and the `'AutoImportFileLog'` argument for the `start` function.

1. Start an SSH session by using PuTTY. Log into the target computer as user `slrt` with password `slrt`. For more information about settings for using PuTTY for an SSH session, see “Execute Target Computer RTOS Commands at Target Computer Command Line”.
2. After you log in, load and run the application to generate file logs. The target computer stores up to the maximum number of logs, in this case 5. At the target computer prompt, type:

   ```
   $ slrealtime load --AppName slrt_ex_osc
   $ slrealtime start
   ```
3. Repeat the previous step until you have created several logs. Between each run, you can change parameter values by loading different parameter set files into the application. For more information, see the `loadParamSet` function.
4. List the logs that you created. At the target computer prompt, type:

   ```
   $ ls applications/slrt_ex_osc/logdata/
   ```

Copy File Logs from Target Computer and Import Folder

1. On the development computer, use a system utility to copy the applications folders from the target computer to an applications folder on the development computer. For example on a Windows computer, you can use `pscp` (a PuTTY utility) or Filezilla. You can download and install PuTTY from www.putty.org. In the MATLAB Command Window, type:

   ```
   system('pscp -r slrt@192.168.7.5:applications C:\work\my_logdata\')
   ```
2. List the file logs that are available to import into the Simulation Data Inspector. In the MATLAB Command Window, type:

   ```
   slrealtime.fileLogList('Directory','C:\work\my_logdata\')
   ```
3. Import the file logs into the Simulation Data Inspector. In the MATLAB Command Window, type:

   ```
   slrealtime.fileLogImport('slrt_ex_osc',...
       'Directory','C:\work\my_logdata\')
   ```

   The simulation runs are available in the Simulation Data Inspector under the Archive list.

Import File Log Data for Selected Run Table

1. After you copy the applications folders from the target computer to an applications folder on the development computer, you can list the file logs that are available to import into the Simulation Data Inspector. With the current folder set to the parent of the applications folder tree, in the MATLAB Command Window, type:

   ```
   my_list = slrealtime.fileLogList()
   ```

   ```
   my_list =
   ```

```
  4×3 table

           Application          StartDate             Size
          _____________    ____________________    _________

    1.    "slrt_ex_osc"    22-Aug-2020 20:10:44    1.2803e+05
    2.    "slrt_ex_osc"    22-Aug-2020 20:11:18    1.2803e+05
    3.    "slrt_ex_osc"    22-Aug-2020 20:11:53    1.2803e+05
    4.    "slrt_ex_osc"    22-Aug-2020 20:12:34    1.2803e+05
```

2 Import the file logs table into the Simulation Data Inspector. In the MATLAB Command Window, type:

```
slrealtime.fileLogImport(my_list)
```

The simulation runs are available in the Simulation Data Inspector.

Import File Log Data for Selected Run

1 After you copy the applications folders from the target computer to an applications folder on the development computer, you can list the file logs that are available to import into the Simulation Data Inspector. With the current folder set to the parent of the `applications` folder, in the MATLAB Command Window, type:

```
slrealtime.fileLogList()
```

```
ans =

  4×3 table

           Application          StartDate             Size
          _____________    ____________________    _________

    1.    "slrt_ex_osc"    22-Aug-2020 20:10:44    1.2803e+05
    2.    "slrt_ex_osc"    22-Aug-2020 20:11:18    1.2803e+05
    3.    "slrt_ex_osc"    22-Aug-2020 20:11:53    1.2803e+05
    4.    "slrt_ex_osc"    22-Aug-2020 20:12:34    1.2803e+05
```

2 Import the file log for a selected run into the Simulation Data Inspector. In the MATLAB Command Window, type:

```
slrealtime.fileLogImport(1)
```

The simulation data for run 1 are available in the Simulation Data Inspector.

Input Arguments

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_number — Run number to import

integer value of an available run

Provides a number for a simulation run file log in the table of available simulation runs.

Example: 1

`run_table` — Run table to import
handle to run table

Provides a handle to a simulation run file log table.

Example: `my_list`

`apps_path` — Path to applications folder
`(fullfile(pwd,'applications'))` (default) | path to applications folder

Provides the path to the applications folder on the development computer to which you have copied the tree of files from the applications folder on the target computer.

Example: `(fullfile(pwd,'applications'))`

Version History

Introduced in R2021a

See Also

`slrealtime.fileLogList` | `'FileLogMaxRuns'` | `'AutoImportFileLog'`

Topics

“Execute Target Computer RTOS Commands at Target Computer Command Line”
“Target Computer Command-Line Interface”
“Save and Reload Parameters by Using the MATLAB Language”

slrealtime.fileLogList

Package: `slrealtime`

After copying file logs from target to development computer, list available file logs for import into Simulation Data Inspector

Syntax

```
slrealtime.fileLogList()
slrealtime.fileLogList('Directory',apps_path)
```

Description

`slrealtime.fileLogList()` lists the available log files for import into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the current folder `pwd`.

Use the `slrealtime.fileLogList` workflow when working with a standalone target computer that does not connect to Simulink Real-Time. If working with a target computer that connects to Simulink Real-Time on the development computer, use the `list` function or **Import File Log** button in the Simulink Real-Time Explorer.

`slrealtime.fileLogList('Directory',apps_path)` lists the available log files for import into the Simulation Data Inspector after you have copied the files from the target computer into the development computer applications folder tree under the selected folder.

Examples

Build and Run Real-Time Application

1. Open model `slrt_ex_osc`.
2. In the Simulink Editor, from the **Real-Time** tab, click **Hardware Settings**.
3. In the **Simulink Real-Time Options** pane, change **Max file log runs** to 5 and click OK.
4. Click **Run on Target**.
5. After the run ends, close the model and exit MATLAB.

Create File Logs on Target Computer

1. Start an SSH session by using PuTTY. Log into the target computer as user slrt with password `slrt`. For more information about settings for using PuTTY for an SSH session, see “Execute Target Computer RTOS Commands at Target Computer Command Line”.
2. After you log in, load and run the application to generate file logs. The target computer stores up to the maximum number of logs, in this case 5. At the target computer prompt, type:

   ```
   $ slrealtime load --AppName slrt_ex_osc
   $ slrealtime start
   ```

3. Repeat the previous step until you have created several logs. Between each run, you can change parameter values by loading different parameter set files into the application. For more information, see the `loadParamSet` function.
4. List the logs that you created. At the target computer prompt, type:

```
$ ls applications/slrt_ex_osc/logdata/
```

Copy File Logs from Target Computer and List Runs

1. On the development computer, use a system utility to copy the applications folders from the target computer to an applications folder on the development computer. For example on a Windows computer, you can use `pscp` (a PuTTY utility) or Filezilla. You can download and install PuTTY from www.putty.org. In the MATLAB Command Window, type:

```
system('pscp -r slrt@192.168.7.5:applications C:\work\my_logdata\')
```

2. List the file logs that are available to import into the Simulation Data Inspector. In the MATLAB Command Window, type:

```
slrealtime.fileLogList('Directory','C:\work\my_logdata\')
```

Input Arguments

`apps_path` — Path to applications folder

`(fullfile(pwd,'applications'))` (default) | path to applications folder

Provides the path to the applications folder on the development computer to which you have copied the tree of files from the applications folder on the target computer.

Example: `(fullfile(pwd,'applications'))`

Version History

Introduced in R2021a

See Also

`slrealtime.fileLogImport`

Topics

“Execute Target Computer RTOS Commands at Target Computer Command Line”
“Target Computer Command-Line Interface”
“Save and Reload Parameters by Using the MATLAB Language”

slrealtime.fmu.compileFMUSources

Package: `slrealtime`

Compile FMU file that contains source code

Syntax

```
slrealtime.fmu.compileFMUSources(fmuFile,Name-Value Arguments)
```

Description

`slrealtime.fmu.compileFMUSources(fmuFile,Name-Value Arguments)` compiles an FMU file that contains source code. The process outputs an FMU file and Simulink Real-Time binary file in the same folder as the input FMU file and appends an `_slrt` suffix to the output file name.

Examples

Compile FMU File and Overwrite Previous Output

This example selects an FMU file to compile and overwrites previous compiler output.

```
% copy an example file to the current working folder
mkdir tempdir myFmuDir
cd tempdir
cd myFmuDir
copyfile(...
    fullfile(matlabroot,'toolbox','slrealtime',...
    'examples','slrt_ex_fmu_work'))
% create variable to provide path and file name
my_file = ['vanDerPol_slrt.fmu']
% compile the FMU file and overwrite previous output
slrealtime.fmu.compileFMUSources(my_file,'overwriteBinary',true)
```

Input Arguments

`fmuFile` — FMU file to compile

`filename`

Selects FMU file to compile. Accepts a file name as input or accepts a variable that provides `[path,filename]` to identify FMU file. If `fmuFile` input is omitted, opens a file selection UI.

Example: `'my_FMU.fmu'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `'overwriteBinary',false,'removeSources',false,'overwriteFMUFile',false`

`overwriteBinary` — Overwrite binary output
`false` (default) | `true`

Selects whether to overwrite an existing binary file in output `binaries` folder.

Example: `'overwriteBinary',false`

`removeSources` — Remove source files
`false` (default) | `true`

Selects whether to remove FMU source files in output `binaries` folder.

Example: `'removeSources',false`

`overwriteFMUFile` — Overwrite FMU file output
`false` (default) | `true`

Selects whether to overwrite FMU file in output `binaries` folder.

Example: `'overwriteFMUFile',false`

Version History

Introduced in R2022a

See Also

Topics
“Apply Functional Mock-up Units by Using Simulink Real-Time”
“Compile Source Code for Functional Mock-up Units”

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.

1. 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);
```

2. View the `Target.ptpd` object `Command` property value.

```
tg.ptpd.Command
```

```
ans =

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

3. View the `Target.ptpd` object `AutoStart` property value.

```
tg.ptpd.AutoStart
```

```
ans =

  logical

   0
```

4. 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
```

Version History

Introduced in R2020b

See Also

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

Topics

“Precision Time Protocol”

“PTP Prerequisites”

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.

1. 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.
2.
```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`stop` | `status` | IEEE 1588 Read Parameter

Topics

“Precision Time Protocol”
“PTP Prerequisites”

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.

1. 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.
2.
```
tg = slrealtime('TargetPC1');
connect(tg);
start(tg.ptpd);
% ... run real-time application
stop(tg.ptpd);
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

`start` | `status` | IEEE 1588 Read Parameter

Topics

“Precision Time Protocol”
“PTP Prerequisites”

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.

1. 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.
2.
```
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 represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

Version History

Introduced in R2020b

See Also

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

Topics

“Precision Time Protocol”
“PTP Prerequisites”

Target.Stimulation

Target computer model root inport stimulator object

Description

A `Target.Stimulation` object represents the stimulation of root inports or Playback blocks of the model running on the target computer.

The object provides access to methods that:

- Start and stop the root inport or Playback block stimulation.
- Pause the root inport or Playback block stimulation.
- Return the status of the root inport or Playback block stimulation.
- Reload the data signal of the root inport or Playback block .

Creation

A `Target.Stimulation` object is created when you create a `Target` object by using the `slrealtime` command. After you create and connect the machine to the `Target` object, you can access the `Target.Stimulation` object. This example creates and connects to `Target` object `tg`, and then starts the stimulation of root inports or Playback blocks on the target computer.

Open the model `slrt_ex_osc_inport` and add an extra inport.

```
model = ('slrt_ex_osc_inport');
open_system(model);
load(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_inport_square.mat'));
waveform = square;
set_param(model,'ExternalInput','waveform');
set_param(model,'LoadExternalInput','on');
set_param(model,'StopTime','Inf');
```

Build the model. Load the real-time application. Start execution. Start stimulation.

```
slbuild(model);
tg = slrealtime('TargetPC1');
connect(tg);
load(tg,model);
start(tg,'StartStimulation','off');
start(tg.Stimulation,'all');
```

Object Functions

getStatus	Return status of root inports stimulation of model on target computer
pause	Pause stimulation of root inports of model on target computer
reloadData	Reload data signal of root inports of model on target computer
start	Start stimulation of root inports of model on target computer
stop	Stop stimulation of root inports of model on target computer

Examples

Start Stimulation of Specific Inports

In the model, start the stimulation of inports named `In1` and `In2`.

```
start(tg.Stimulation,{'In1','In2'});
% if the port number of inport named 'In1' is 1
% and the port number of inport named 'In2' is 2
% this syntax is equivalent to:
%
% start(tg.Stimulation,[1,2]);
```

Pause Stimulation of Specific Inports

In the model, pause the stimulation of inports named `In1` and `In2`.

```
pause(tg.Stimulation,{'In1','In2'});
% this syntax is equivalent to:
% pause(tg.Stimulation,[1,2]);
```

Stop Stimulation of Specific Inports

In the model, stop the stimulation of inports named `In1` and `In2`.

```
stop(tg.Stimulation,{'In1','In2'});
% this syntax is equivalent to:
% stop(tg.Stimulation,[1,2]);
```

Version History

Introduced in R2021a

See Also

`Target` | `stop` | `getStatus` | `reloadData` | `pause` | `start` | Playback

Topics

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

getStatus

Package: `slrealtime`

Return status of root inports stimulation of model on target computer

Syntax

```
getStatus(target_object.Stimulation,inports)
getStatus(target_object.Stimulation,playbacks)
getStatus(target_object.Stimulation,'all')
```

Description

`getStatus(target_object.Stimulation,inports)` returns the status of the stimulation of specified root inports of the model running on the target computer. The status of the stimulation can be:

- `RUNNING` — indicates that stimulation is running
- `PAUSED` — indicates that stimulation is paused
- `isFinished` — indicates that stimulation data is finished
- `STOPPED` — indicates that stimulation is stopped

`getStatus(target_object.Stimulation,playbacks)` returns the status of the stimulation of specified Playback blocks in the model running on the target computer. The status of the stimulation can be `RUNNING`, `PAUSED`, `isFinished`, or `STOPPED`.

`getStatus(target_object.Stimulation,'all')` returns the status of the stimulation of all root inports of the model and Playback blocks in the model running on the target computer.

Examples

Return Stimulation Status of Specific Inports

Get the status of stimulation of inports named `first` and `third`.

```
status = getStatus(tg.Stimulation,{'first','third'});
% if the port number of inport named 'first' is 1
% and the port number of inport named 'third' is 3
% this syntax is equivalent to:
%
% status = getStatus(tg.Stimulation,[1,3]);
```

```
status =

  struct with fields:

    first: RUNNING
    third: RUNNING
```

Return Stimulation Status of Specific Playback Blocks

Get the status of stimulation of Playback blocks named `first` and `third`.

```
status = getStatus(tg.Stimulation,{'first','third'});
```

```
status =

  struct with fields:

    first: RUNNING
    third: RUNNING
```

Return Stimulation Status of All Inports and Playback Blocks

Get the status of stimulation of all inports and Playback blocks.

```
tg.Stimulation.getStatus('all');
```

Input Arguments

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

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`inports` — Specific inports of the model on target computer
array of inport numbers | cell array of inport names | cell array of inport block paths

Specifies the numbers of the inports or names of the inports or block paths of the inports present on the model running on the target computer.

Example: `[1,3,5]`, `{'in1','in2'}`, `{'model_name/in1','model_name/in4'}`

`playbacks` — Specific Playback blocks in the model on target computer
cell array of Playback block names | cell array of Playback block paths

Specifies the block names or block paths of the Playback blocks present in the model running on the target computer. If you use a block name that is not unique (more than one block with that name is present) in the model, the function returns an error and suggests using block paths instead.

Example: `{'pb1','pb2'}`, `{'model_name/pb1','model_name/pb4'}`

`all` — All the root inports of the model on target computer
'all'

Represents all the available root inports of the model running on the target computer.

Example: `'all'`

Version History

Introduced in R2021a

See Also

`Target` | `Target.Stimulation` | `stop` | `start` | `reloadData` | `pause`

Topics

“Stimulate Root Inport by Using MATLAB Language”
“Signal Logging and Streaming Basics”

pause

Package: `slrealtime`

Pause stimulation of root inports of model on target computer

Syntax

```
pause(target_object.Stimulation, inports)
pause(target_object.Stimulation, playbacks)
pause(target_object.Stimulation,'all')
```

Description

`pause(target_object.Stimulation, inports)` pauses the stimulation of the specified root inports of the model running on the target computer.

`pause(target_object.Stimulation, playbacks)` pauses the stimulation of the specified Playback blocks in the model running on the target computer.

`pause(target_object.Stimulation,'all')` pauses the stimulation of all root inports of the model and Playback blocks in the model running on the target computer.

Examples

Pause Stimulation of Specific Inports

In a model with five inports, pause the stimulation of inports named `first` and `third`.

```
pause(tg.Stimulation,{'first','third'});
% if the port number of inport named 'first' is 1
% and the port number of inport named 'third' is 3
% this syntax is equivalent to:
%
% pause(tg.Stimulation,[1,3]);
```

Pause Stimulation of Specific Playback Blocks

In a model with five Playback blocks, pause the stimulation of Playback blocks named `first` and `third`.

```
pause(tg.Stimulation,{'first','third'});
```

Pause Stimulation of All Inports and Playback blocks

In a model with five inports, pause the stimulation of all inports and Playback blocks.

```
tg.Stimulation.pause('all');
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`inports` — Specific inports of the model on target computer

array of inport numbers | cell array of inport names | cell array of inport block paths

Specifies the numbers of the inports or names of the inports or block paths of the inports present on the model running on the target computer.

Example: `[1,3,5]`, `{'in1','in2'}`, `{'model_name/in1','model_name/in4'}`

`playbacks` — Specific Playback blocks in the model on target computer

cell array of Playback block names | cell array of Playback block paths

Specifies the block names or block paths of the Playback blocks present in the model running on the target computer. If you use a block name that is not unique (more than one block with that name is present) in the model, the function returns an error and suggests using block paths instead.

Example: `{'pb1','pb2'}`, `{'model_name/pb1','model_name/pb4'}`

`all` — All the root inports of the model on target computer

'all'

Represents all the available root inports of the model running on the target computer.

Example: `'all'`

Version History

Introduced in R2021a

See Also

`Target` | `Target.Stimulation` | `stop` | `start` | `getStatus` | `reloadData`

Topics

“Stimulate Root Inport by Using MATLAB Language”
“Signal Logging and Streaming Basics”

reloadData

Package: `slrealtime`

Reload data signal of root inports of model on target computer

Syntax

```
reloadData(target_object.Stimulation, inport, tsName)
reloadData(target_object.Stimulation, playback, loadSource)
```

Description

`reloadData(target_object.Stimulation, inport, tsName)` reloads the data signal from a `timeseries` object to the specified root inport of the model running on the target computer.

`reloadData(target_object.Stimulation, playback, loadSource)` reloads the data signal from a `timeseries` object, a `Simulink.SimulationData.Dataset` object, or pairs of ports and `timeseries` objects for the specified Playback block in the model running on the target computer.

Examples

Reload Inport Data

To load data to an inport, create a time series object.

1.
```
sampleTime = 0.1; %sample time of the model
endTime = 10; %end time of the model
numberOfSamples = endTime * 1/sampleTime + 1;
timeVector = (0:numberOfSamples) * sampleTime;
u = timeseries(timeVector*10,timeVector);
```
2. Load the object to the inport named `first`.

```
reloadData(tg.Stimulation,'first',u);
```
3. To load the same object to multiple inports named `first` and `third`.

```
reloadData(tg.Stimulation,'first',u,'third',u);
```

Reload Playback Block Data

To load data to a Playback block, create a time series object.

1.
```
sampleTime = 0.1; %sample time of the model
endTime = 10; %end time of the model
numberOfSamples = endTime * 1/sampleTime + 1;
timeVector = (0:numberOfSamples) * sampleTime;
u = timeseries(timeVector*10,timeVector);
```
2. Load the object to the Playback block named `first`.

```
reloadData(tg.Stimulation,'first',u);
```

3 To load the objects `playbackData1` and `playbackData2` to a Playback blocks named `first` and its ports 1 and 2.

```
reloadData(tg.Stimulation,'first',{{1,playbackData1},{2,playbackData2}});
```

Input Arguments

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

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`inport` — Specific inport of the model on target computer
inport name | inport number | block path of inport

Specifies the name of the inport or inport number or block path of the inport present on the model running on the target computer.

Example: `{'signal_1'}, [2], {'model_name/in4'}`

`tsName` — Time series object to load data to the inport
`timeseries` object

Specifies a `timeseries` object to load into the inports.

Example: `u`

`playback` — Specific Playback block in the model on target computer
block name | block path of Playback block

Specifies the name of the Playback block or block path of the Playback block present in the model running on the target computer.

Example: `{'signal_1'}, [2], {'model_name/in4'}`

`loadSource` — Source object for load data into the Playback block
`timeseries` object | `Simulink.SimulationData.Dataset` object | cell array of pairs of Playback block port numbers and `timeseries` objects

Specifies a `timeseries` object or `Simulink.SimulationData.Dataset` object to load into the Playback block. Alternatively, specifies pairs of port numbers and `timeseries` object to load into the Playback block.

Example: `{{1,playbackData1},{2,playbackData2}}`

Version History

Introduced in R2021a

See Also

`Target` | `Target.Stimulation` | `stop` | `start` | `pause` | `getStatus`

Topics
“Stimulate Root Inport by Using MATLAB Language”

“Signal Logging and Streaming Basics”

start

Package: `slrealtime`

Start stimulation of root inports of model on target computer

Syntax

```
start(target_object.Stimulation,inports)
start(target_object.Stimulation,playbacks)
start(target_object.Stimulation,'all')
```

Description

`start(target_object.Stimulation,inports)` starts the stimulation of specified root inports of the model running on the target computer.

`start(target_object.Stimulation,playbacks)` starts the stimulation of specified Playback blocks in the model running on the target computer.

`start(target_object.Stimulation,'all')` starts the stimulation of all root inports of the model and Playback blocks in the model running on the target computer.

Examples

Start Stimulation of Specific Inports

In a model with five inports, start the stimulation of inports named `first` and `third`.

```
start(tg.Stimulation,{'first','third'});
% if the port number of inport named 'first' is 1
% and the port number of inport named 'third' is 3
% this syntax is equivalent to:
%
% start(tg.Stimulation,[1,3]);
```

Start Stimulation of Specific Playback Blocks

In a model with five Playback blocks, start the stimulation of Playback blocks named `playThis` and `playThat`.

```
start(tg.Stimulation,{'playThis','playThat'});
```

Start Stimulation of All Inports

In a model with five inports, start the stimulation of all inports.

```
tg.Stimulation.start('all');
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`inports` — Specific inports of the model on target computer

array of inport numbers | cell array of inport names | cell array of inport block paths

Specifies the numbers of the inports or names of the inports or block paths of the inports present on the model running on the target computer.

Example: `[1,3,5]`, `{'in1','in2'}`, `{'model_name/in1','model_name/in4'}`

`playbacks` — Specific Playback blocks in the model on target computer

cell array of Playback block names | cell array of Playback block paths

Specifies the block names or block paths of the Playback blocks present in the model running on the target computer. If you use a block name that is not unique (more than one block with that name is present) in the model, the function returns an error and suggests using block paths instead.

Example: `{'pb1','pb2'}`, `{'model_name/pb1','model_name/pb4'}`

`all` — All the root inports of the model on target computer

'all'

Represents all the available root inports of the model running on the target computer.

Example: `'all'`

Version History

Introduced in R2021a

See Also

`Target` | `Target.Stimulation` | `stop` | `getStatus` | `reloadData` | `pause`

Topics

“Stimulate Root Inport by Using MATLAB Language”
“Signal Logging and Streaming Basics”

stop

Package: `slrealtime`

Stop stimulation of root inports of model on target computer

Syntax

```
stop(target_object.Stimulation,inports)
stop(target_object.Stimulation,playbacks)
stop(target_object.Stimulation,'all')
```

Description

`stop(target_object.Stimulation,inports)` stops the stimulation of the specified root inports of the model running on the target computer.

`stop(target_object.Stimulation,playbacks)` stops the stimulation of the specified Playback blocks in the model running on the target computer.

`stop(target_object.Stimulation,'all')` stops the stimulation of all root inports of the model and Playback blocks in the model running on the target computer.

Examples

Stop Stimulation of Specific Inports

In a model with five inports, stop the stimulation of inports named `first` and `third`.

```
stop(tg.Stimulation,{'first','third'});
% if the port number of inport named 'first' is 1
% and the port number of inport named 'third' is 3
% this syntax is equivalent to:
%
% stop(tg.Stimulation,[1,3]);
```

Stop Stimulation of Specific Playback Blocks

In a model with five Playback blocks, stop the stimulation of Playback blocks named `first` and `third`.

```
stop(tg.Stimulation,{'first','third'});
```

Stop Stimulation of All Inports and Playback blocks

In a model with five inports, stop the stimulation of all inports and Playback blocks.

```
tg.Stimulation.stop('all');
```

Input Arguments

`target_object` — Object that represents target computer

`slrealtime.Target` object

Provides access to methods that manipulate the target computer properties.

Example: `tg`

`inports` — Specific inports of the model on target computer

array of inport numbers | cell array of inport names | cell array of inport block paths

Specifies the numbers of the inports or names of the inports or block paths of the inports present on the model running on the target computer.

Example: `[1,3,5]`, `{'in1','in2'}`, `{'model_name/in1','model_name/in4'}`

`playbacks` — Specific Playback blocks in the model on target computer

cell array of Playback block names | cell array of Playback block paths

Specifies the block names or block paths of the Playback blocks present in the model running on the target computer. If you use a block name that is not unique (more than one block with that name is present) in the model, the function returns an error and suggests using block paths instead.

Example: `{'pb1','pb2'}`, `{'model_name/pb1','model_name/pb4'}`

`all` — All the root inports of the model on target computer

'all'

Represents all the available root inports of the model running on the target computer.

Example: `'all'`

Version History

Introduced in R2021a

See Also

`Target` | `Target.Stimulation` | `start` | `getStatus` | `reloadData` | `pause`

Topics

“Stimulate Root Inport by Using MATLAB Language”
“Signal Logging and Streaming Basics”

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 1-163

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.

1 Create targets object `my_tgs`. Add target computers to the targets object. Assign target computers to target objects. Create a 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'
```

2 Set `Target` object `tg1` IP address to `'192.168.7.5'` by using the `TargetSettings` property.

```
tg1.TargetSettings.address = '192.168.7.5';
tg1.TargetSettings;
```

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

Change Password for Target Computer

For security, some installations require changing the default `userPassword` for the target computer. To customize the password, change both:

— The `userPassword` in the `TargetSettings`

— The password for the `slrt` user on the corresponding target computer

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

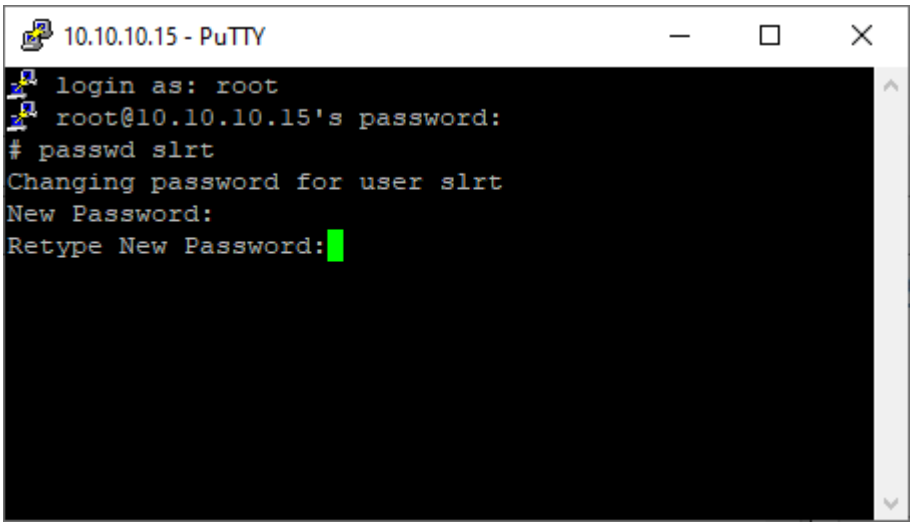
```
my_tgs = slrealtime.Targets();
% do not need to add default target 'TargetPC1'
addTarget(my_tgs,'TargetPC2');
addTarget(my_tgs,'TargetPC3');

% 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'
```

2 Set `Target` object `tg1` `userPassword` to `'H3lloThere!'` by using the `TargetSettings` property.

```
tg1.TargetSettings.userPassword = 'H3lloThere!';
tg1.TargetSettings;
```

To set the password on the target computer, open a PuTTY session to the target computer (log in as user `root` and password `root`) and use the `passwd` command to set the password for the

`slrt`. For more information about using PuTTY, see “Execute Target Computer RTOS Commands at Target Computer Command Line”.



Version History

Introduced in R2020b

See Also

`addTarget` | `removeTarget` | `getTargetSettings`

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`

Version History

Introduced in R2020b

See Also

`Targets` | `removeTarget` | `getTargetSettings`

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.

1
```
my_tgs = slrealtime.Targets();
my_tgs_settings = getTargetSettings(my_tgs)
```

```
my_tgs_settings =

  TargetSettings with properties:

           name: 'TargetPC1'
        address: '192.168.7.5'
        sshPort: 22
        xcpPort: 5555
       username: 'slrt'
   userPassword: 'slrt'
   rootPassword: 'root'
```

2 Get target computer name properties from `Targets` object.

```
my_tgs_settings.name
```

```
ans =

    'TargetPC1'


ans =

    'TargetPC2'
```

3 Get target computer address properties from `Targets` object.

```
my_tgs_settings.address
```

```
ans =
```

```
        '192.168.7.5'

    ans =

        '192.168.7.10'
```

4 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`

Version History

Introduced in R2020b

See Also

`Targets` | `addTarget` | `removeTarget`

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`

Version History

Introduced in R2020b

See Also

`Targets` | `addTarget` | `getTargetSettings`

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`

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

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 the object 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 1-175

Example: “Update Root-Level Inport Data” on page 1-175

Example: “Get and Set Application Options” on page 1-175

Example: “Get Application Signals and Parameters” on page 1-176

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 an example, see “Get and Set Application Options” on page 1-175. 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`.
- `overrideBaseRatePeriod` selects an override value for the application base rate period.
- `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.
- `startupParameterSet` indicates the start up parameter set from the `ParameterSet` objects that have been added to the `Application` object. To change the selection, use the `updateStartupParameterSet` function.
- `stoptime` selects the stop time for the real-time application.

Object Functions

addParamSet	Add a parameter set to a real-time application
extractASAP2	Extract generated A2L file from real-time application file
getAllFileLogBlocks	Returns block paths corresponding to File Log blocks in application
getFileLogDecimation	Returns decimation value of File Log block based on block path
getInformation	Get real-time application information
getParameters	Get real-time application parameters
getRootLevelInports	Returns root level inports in application
getSignals	Get real-time application signals
setFileLogDecimation	Sets decimation value on File Log blocks based on block path and input decimation value
updateASAP2	Pack the ASAP2 file into application

updateRootLevelInportData	Replace external input data in real-time application with input data
updateAutoSaveParameterSetOnStop	Update the auto save parameter set on stop for an application
updateStartupParameterSet	Update the startup parameter set for an application

Examples

Extract ASAP2 File

Retrieve the ASAP2 file from real-time application.

1. Create an application object for the real-time application.

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

2. Retrieve the ASAP2 file from the real-time application.

```
extractASAP2(app_obj);
```

Update Root-Level Inport Data

Change waveform data from square wave to sine wave.

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

```
waveform = sinewave;
```

2. Create an application object.

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

3. Update the inport data.

```
updateRootLevelInportData(app_object)
```

4. 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.

1. Create an application object.

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

2. View application options by getting the application Options property values.

```
my_app.Options.get
```

```
ans =

  struct with fields:
```

```
          fileLogMaxRuns: 1
                 loglevel: "info"
    overrideBaseRatePeriod: 0
          pollingThreshold: 1.0000e-04
       startupParameterSet: "paramInfo"
                 stoptime: Inf
```

3 Change the application stop time value option.

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

Note You can inadvertently delete existing file logs for an installed real-time application on the target computer if you use the `slrealtime.Application` function to change the Options for `FileLogMaxRuns` and then reload the application. To change the number of stored logs without deleting existing logs, load the real-time application and then change the FileLogMaxRuns option by using the `start(tg)` function.

4 Save application options to a MATLAB variable. Apply options from the 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.

1 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]
```

2 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
```

3 View application signals as array elements.

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

```
ans =

    'slrt_ex_param_tuning/Gain'
```

4 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
```

5 View application parameters as array elements.

```
my_params(1).BlockParameterName

ans =

    'Gain'
```

Version History

Introduced in R2020b

R2022b: Added Application Parameter Set Options and Auto Save

In R2022b, you can get the startup parameter set for an `Application` object by using the `startupParameterSet` value in the `Options` property. You can update the parameter set selection by using the `updateStartupParameterSet` function. You can enable auto save of the parameter set by using the `updateAutoSaveParameterSetOnStop` function.

See Also

`extractASAP2` | `getInformation` | `getParameters` | `getSignals` | `updateRootLevelInportData`

Topics

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

addParamSet

Package: `slrealtime`

Add a parameter set to a real-time application

Syntax

```
addParamSet(app_object,parameter_set)
```

Description

`addParamSet(app_object,parameter_set)` adds a `ParameterSet` object to a real-time application MLDATX on the development computer. When the real-time application is loaded or installed on the target computer, the parameter sets added to the application appear on the target computer for the application.

Examples

Add Parameter Set to Application

To add a `ParameterSet` object to a real-time application, use the `addParamSet` function.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
load(tg,mdlName);
paramSetName = 'outportTypes';
saveParamSet(tg,paramSetName);
myParamSet = importParamSet(tg,paramSetName);
addParamSet(app_object,myParamSet);
```

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.

`parameter_set` — `ParameterSet` object

`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

Version History

Introduced in R2021a

See Also

`importParamSet` | `saveParamSet` | `updateStartupParameterSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

extractASAP2

Package: `slrealtime`

Extract generated A2L file from real-time application file

Syntax

```
extractASAP2(app_object)
extractASAP2(app_object,Name,Value)
```

Description

`extractASAP2(app_object)` retrieves an A2L file from a real-time application file and save the file in the working folder. The A2L file can only be extracted if it was previously packaged into the real-time application `MLDATX` file using `updateASAP2` method.

`extractASAP2(app_object,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.

```
% build mymodel, generate the a2l file, and copy it into the current folder
slbuild('mymodel');
coder.asap2.export('mymodel','MapFile','mymodel_slrealtime_rtw/mymodel','Comments',false);
copyfile(fullfile(pwd,'/mymodel_slrealtime_rtw/mymodel.a2l'));

% replace the a2l file in the application
app_obj = slrealtime.Application('mymodel.mldatx')
updateASAP2(app_obj,'mymodel.a2l')

% extract a2l file from mymodel application file
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')
myFolder = fullfile(userpath,'temp')
extractASAP2(app_obj,'Folder',myFolder)
```

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_object` — Object that represents 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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FileName', 'CustomName', 'Folder', myFolder`

`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', myFolder`

`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'`

Version History

Introduced in R2020b

See Also

`updateRootLevelInportData` | `Application` | `updateASAP2`

getAllFileLogBlocks

Package: `slrealtime`

Returns block paths corresponding to File Log blocks in application

Syntax

```
allFileLogBlocks = getAllFileLogBlocks(app_object)
```

Description

`allFileLogBlocks = getAllFileLogBlocks(app_object)` returns a cell array of the block paths. These block paths corresponding to the File Log blocks in the real-time application. You can use the returned cell array to get or set the decimation value of the File Log blocks.

Examples

Get File Log Blocks in Application

Use the `getAllFileLogBlocks` function to get the block paths of File Log blocks in a real-time application object.

```
my_App = slrealtime.Application('slrt_ex_osc');
myFileLogBlocks = getAllFileLogBlocks(my_App)
```

```
myFileLogBlocks =

  2×1 cell array

    {["slrt_ex_osc/File Log"]}
    {["slrt_ex_osc/File Log"]}
```

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

`allFileLogBlocks` — Returned cell array of block paths

cell array of block paths

The `allFileLogBlocks` returned by the `getAllFileLogBlocks` function contains the block paths of File Log blocks in the real-time application.

Version History

Introduced in R2022a

See Also

`Application` | `getFileLogDecimation` | `setFileLogDecimation`

getFileLogDecimation

Package: `slrealtime`

Returns decimation value of File Log block based on block path

Syntax

```
fileLogDecimation = getFileLogDecimation(app_object,block_path)
```

Description

`fileLogDecimation = getFileLogDecimation(app_object,block_path)` returns the value of decimation for the specified File Log block. You can use this function to get the decimation setting of a particular File Log block or to verify the changed decimation after using the `setFileLogDecimation` function.

Examples

Get Decimation Value for File Log Block

Use the `getFileLogDecimation` function to get the decimation setting for a File Log block that you specify by using a block paths from a real-time application object.

1 Get the block paths for File Log blocks in the real-time application.

```
myApp = slrealtime.Application('slrt_ex_osc');
myFileLogBlocks = getAllFileLogBlocks(my_App)
```

```
myFileLogBlocks =

  2×1 cell array

    {["slrt_ex_osc/File Log"]}
    {["slrt_ex_osc/File Log"]}
```

2 Get the decimation value for a selected File Log block.

```
myDecimation = getFileLogDecimation(myApp,"slrt_ex_osc/File Log")
```

```
myDecimation =

  int32

   1
```

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.

`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'}`

Output Arguments

`fileLogDecimation` — Decimation setting of File Log block
int32

The returned `fileLogDecimation` is the decimation value of the selected File Log block.

Version History

Introduced in R2022a

See Also

`Application` | `getAllFileLogBlocks` | `setFileLogDecimation`

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.

1 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]
```

2 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'
```

3 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

Version History

Introduced in R2020b

See Also

`Target` | `Application` | `getSignals`

Topics

“Add App Designer App to Inverted Pendulum Model”

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.

1 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]
```

2 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
```

3 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

Version History

Introduced in R2020b

See Also

`Target` | `Application` | `getSignals`

Topics
"Add App Designer App to Inverted Pendulum Model"

getRootLevelInports

Package: `slrealtime`

Returns root level inports in application

Syntax

```
rootInports = getRootLevelInports(app_object)
```

Description

`rootInports = getRootLevelInports(app_object)` returns root level inports in the application as a cell array.

Examples

Get Root Level Inports in Application

Use the `getRootLevelInports` function to get the root level inports in a real-time application object.

```
myApp = slrealtime.Application('slrt_ex_osc_inport');
myRoots = getRootLevelInports(myApp)
```

```
myRoots =

     []
```

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

`rootInports` — Returned cell array of root inports

cell array of root inports

The `rootInports` returned by the `getRootLevelInports` function contains the root inports in the real-time application.

Version History

Introduced in R2022a

See Also

`Application` | `updateRootLevelInportData`

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.

1 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]
```

2 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
```

3 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

Version History

Introduced in R2020b

See Also

`Target` | `Application` | `getParameters`

Topics
“Add App Designer App to Inverted Pendulum Model”

setFileLogDecimation

Package: `slrealtime`

Sets decimation value on File Log blocks based on block path and input decimation value

Syntax

```
setFileLogDecimation(app_object,block_path,decimationValue)
```

Description

`setFileLogDecimation(app_object,block_path,decimationValue)` sets the decimation value for the specified File Log block. This function modifies the decimation value of the File Log block in the application object. The subsequent loads of the real-time application MLDATX file run on the target with the modified decimation value of the File Log block.

Examples

Set Decimation Value for File Log Block

Use the `setFileLogDecimation` function to set the decimation setting for a File Log block that you specify by using a block paths from a real-time application object.

1 Get the block paths for File Log blocks in the real-time application.

```
myApp = slrealtime.Application('slrt_ex_osc');
myFileLogBlocks = getAllFileLogBlocks(my_App)
```

```
myFileLogBlocks =

  2×1 cell array

    {["slrt_ex_osc/File Log"]}
    {["slrt_ex_osc/File Log"]}
```

2 Get the decimation value for a selected File Log block.

```
myDecimation = getFileLogDecimation(myApp,"slrt_ex_osc/File Log")
```

```
myDecimation =

  int32

   1
```

3 Set the updated decimation value for a selected File Log block.

```
setFileLogDecimation(myApp,"slrt_ex_osc/File Log",2)
```

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.

`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'}`

`decimationValue` — Decimation value for File Log block
int32

The `decimationValue` provides the updated decimation value for the File Log block.

Example: 2

Data Types: `int32`

Version History

Introduced in R2022a

See Also

`Application` | `getAllFileLogBlocks` | `getFileLogDecimation`

updateASAP2

Package: `slrealtime`

Pack the ASAP2 file into application

Syntax

```
updateASAP2(app_object,updated_a2l)
```

Description

`updateASAP2(app_object,updated_a2l)` packs the `updated_a2l` file into the `app_object` application by replacing the ASAP2 file present in the application.

Examples

Pack ASAP2 File to Simulink Real-Time Application

Replace the ASAP2 file from the application with the customized ASAP2 file.

```
% replace the a2l file in the application
app_obj = slrealtime.Application('mymodel.mldatx')
updateASAP2(app_obj,'my_updated_a2l.a2l')
```

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.

`updated_a2l` — New ASAP2 file

char vector | string

Specify the name of the customized ASAP2 file to pack it into the Simulink Real-Time application.

Example: `new_asap2`

Version History

Introduced in R2022b

See Also

`extractASAP2` | `Application`

updateAutoSaveParameterSetOnStop

Package: `slrealtime`

Update the auto save parameter set on stop for an application

Syntax

```
updateAutoSaveParameterSetOnStop(app_object,boolean)
```

Description

`updateAutoSaveParameterSetOnStop(app_object,boolean)` enables autos save of the real-time application current parameter set to the target computer when the application stops.

Examples

Update Auto Save Parameter Set On Stop for Application

To update the auto save parameter set on stop for a real-time application from a `ParameterSet` object, use the `updateAutoSaveParameterSetOnStop` function.

```
% create and import a parameter set
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
load(tg,mdlName);
paramSetName = 'outportTypes';
saveParamSet(tg,paramSetName);
myParamSet = importParamSet(tg,paramSetName);

% modify parameter set value in parameter set
set(myParamSet,'slrt_ex_osc_outport/Signal Generator','Amplitude',10);

% add parameter set into real-time application
% and set as startup parameter set
myApp = slrealtime.Application(mdlName);
updateAutoSaveParameterSetOnStop(myApp,true);
```

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.

`boolean` — logical value

`true` | `false`

Enable auto save with value `true`. Disable auto save with value `false`.

Example: `true`

Data Types: `logical`

Version History

Introduced in R2022b

See Also

`importParamSet` | `saveParamSet` | `addParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

updateStartupParameterSet

Package: `slrealtime`

Update the startup parameter set for an application

Syntax

```
updateStartupParameterSet(app_object,filename)
```

Description

`updateStartupParameterSet(app_object,filename)` updates the selection of the startup parameter set for a real-time application from a parameter set file on the target computer. After adding one or more `ParameterSet` objects to an application by using the `addParamSet` function, you can choose which of these parameter sets is loaded into the real-time application on startup by using the `updateStartupParameterSet` function.

Examples

Update Startup Parameter Set for Application

To update the startup parameter set for a real-time application from a `ParameterSet` object, use the `updateStartupParameterSet` function.

```
% create and import a parameter set
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
load(tg,mdlName);
paramSetName = 'outportTypes';
saveParamSet(tg,paramSetName);
myParamSet = importParamSet(tg,paramSetName);

% modify parameter set value in parameter set
set(myParamSet,'slrt_ex_osc_outport/Signal Generator','Amplitude',10);

% add parameter set into real-time application
% and set as startup parameter set
myApp = slrealtime.Application(mdlName);
addParamset(myApp,myParamSet);
updateStartupParameterSet(myApp,paramSetName);

% load real-time application and
% check that modified parameter set is loaded
load(tg,mdlName);
getparam(tg,'slrt_ex_osc_outport/Signal Generator','Amplitude')
```

```
ans =

     10
```

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.

`filename` — Name of a parameter set file on the target computer

character vector | string scalar

Enter the name of the parameter set file from the target computer file system.

Example: `'outportTypes'`

Data Types: `char` | `string`

Version History

Introduced in R2021a

See Also

`importParamSet` | `saveParamSet` | `addParamSet` | `Application` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

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. For a more detailed example, see “Define and Update Inport Data”.

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

   ```
   waveform = sinewave;
   ```
2. Create an application object.

   ```
   app_object = slrealtime.Application('slrt_ex_slrt_osc_inport');
   ```
3. Update inport data.

   ```
   updateRootLevelInportData(app_object)
   ```
4. 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.

Version History

Introduced in R2020b

See Also

`Target` | `Application`

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

ParameterSet

Real-time application parameter set

Description

A `ParameterSet` object represents the contents of a parameter set file imported from a real-time application that is loaded on a target computer and provides access to methods and properties related to the parameter set file.

The object provides access to methods and properties that:

- Save parameters from a real-time application to a parameter set file.
- Import parameter set file data into a `ParameterSet` object.
- Tune parameters in the real-time application by using the `ParameterSet` object.
- Apply the tuned parameters from the real-time application to the model.

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

Create a `ParameterSet` object by using the `importParamSet` command. After you create and connect to the `Target` object and load the real-time application on the target computer, you import the parameter set information from the loaded application into a `ParameterSet` object. This example creates and connects to `Target` object `tg`, loads a real-time application, creates a parameter set file, and imports parameter set information into a `ParameterSet` object `myParamSet` on the development computer.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
connect(tg);
load(tg,mdlName);
paramSetName = 'myParamSet';
saveParamSet(tg,paramSetName);
myParamSet = importParamSet(tg,paramSetName);
```

Properties

`filename` — file name for parameter set
character vector | string

The `filename` property holds the parameter set file name on the target computer. This property is set by using the `saveParamSet` method.

Example: `'myParamSet'`

Object Functions

delete	Deletes a ParameterSet object
explorer	Open Parameter Explorer and view Parameter Set
exportToModel	Export values from ParameterSet object to model
set	Set a parameter value in a ParameterSet object
syncWithApp	Sync model parameters to real-time application parameters

Examples

Tune Parameters by Using Parameter Set Object

The `ParameterSet` object and methods let you tune parameters in the real-time application and apply the tuned parameters to the model. For a flowchart of this workflow, see “Save and Reload Parameters by Using the MATLAB Language”.

1 Build the model and load the real-time application.

```
mdlName = 'slrt_ex_osc_outport';
slbuild(mdlName);
tg = slrealtime('TargetPC1');
load(tg,mdlName);
```

2 Save the parameter set to a file.

```
paramSetName = 'outportTypes';
saveParamSet(tg,paramSetName);
```

3 Import the parameter set into a `ParameterSet` object on the development computer.

```
myParamSet = importParamSet(tg,paramSetName);
```

4 To view or edit the parameters, open the `ParameterSet` object in the Simulink Real-Time Parameter Explorer UI.

```
explorer(myParamSet);
```

5 After tuning the parameters, export the modified parameter set to the parameter set file on the target computer and load the parameters into the real-time application.

```
exportParamSet(tg,myParamSet);
loadParamSet(tg,myParamSet.filename);
```

6 To synchronize the parameter name-value pairs and synchronize the model checksum saved in the parameter set object with the real-time application, use the `syncWithApp` command.

```
syncWithApp(myParamSet,mdlName);
```

Set a Parameter

1 To set a parameter value in the `ParameterSet` object programmatically instead of using the Simulink Real-Time Parameter Explorer UI, use the `set` command.

2
```
set(myParamSet,'slrt_ex_osc_outport/Signal Generator','Amplitude',2);
```

Delete a Parameter Set

1. To delete the contents of a `ParameterSet` object, use the `delete` command.
2. `delete(myParamSet);`

Version History

Introduced in R2021a

See Also

`exportParamSet` | `deleteParamSet` | `getparam` | `getParameters` | `importParamSet` | `listParamSet` | `loadParamSet` | `saveParamSet` | `setparam` | `addParamSet` | `updateStartupParameterSet` | `Application` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”
“Troubleshoot Instance-Specific Parameters Not Saved”

delete

Package: `slrealtime`

Deletes a `ParameterSet` object

Syntax

```
delete(parameter_set)
```

Description

`delete(parameter_set)` deletes the contents of a `ParameterSet` object.

Examples

Delete Content of Parameter Set Object

To delete the contents of a `ParameterSet` object, use the delete function.

```
delete(myParamSet)
```

Input Arguments

`parameter_set` — `ParameterSet` object
`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

Version History

Introduced in R2021a

See Also

`listParamSet` | `ParameterSet` | `Target`

Topics
“Save and Reload Parameters by Using the MATLAB Language”

explorer

Package: `slrealtime`

Open Parameter Explorer and view Parameter Set

Syntax

```
explorer(parameter_set)
```

Description

`explorer(parameter_set)` opens the Simulink Real-Time Parameter Explorer and loads the `ParameterSet` object.

Examples

Open Parameter Explorer

Open the Parameter Explorer and view the parameter set.

```
explorer(myParamSet)
```

Input Arguments

`parameter_set` — `ParameterSet` object

`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

Version History

Introduced in R2021a

See Also

Topics

“Save and Reload Parameters by Using the MATLAB Language”

exportToModel

Package: `slrealtime`

Export values from `ParameterSet` object to model

Syntax

```
exportToModel(parameter_set,model_name)
```

Description

`exportToModel(parameter_set,model_name)` exports the parameter values from the `ParameterSet` object into the model.

Examples

Export Values from Parameter Set into Model

To export the parameter set values from the `ParameterSet` object into the model, use the `exportToModel` function.

```
exportToModel(myParamSet,'slrt_ex_osc_outport')
```

Input Arguments

`parameter_set` — `ParameterSet` object
`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

`model_name` — Simulink model name
character vector | string scalar

Provides the name of a model to which the parameter values are exported. The model must be the same model that built the real-time application MLDATX file from which the `ParameterSet` object was created.

Example: `'slrt_ex_osc'`

Version History

Introduced in R2021a

See Also

`explorer` | `ParameterSet` | `Target`

Topics
“Save and Reload Parameters by Using the MATLAB Language”

set

Package: `slrealtime`

Set a parameter value in a `ParameterSet` object

Syntax

```
set(parameter_set,block_path,parameter_name,parameter_value)
```

Description

`set(parameter_set,block_path,parameter_name,parameter_value)` provides a programmatic approach that performs the same operation as editing the value in the Parameter Explorer. For more information, see `explorer`.

Examples

Set Parameter Value in Parameter Set Object

To set a parameter value in the `ParameterSet` object, use the `set` command.

```
set(myParamSet,'slrt_ex_osc_outport/Signal Generator/Amplitude',2);
```

Input Arguments

`parameter_set` — `ParameterSet` object
`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

`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'`, `'G2'`

`parameter_value` — value of the parameter
parameter value

The value of the parameter.

Version History

Introduced in R2021a

See Also

`explorer` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

syncWithApp

Package: `slrealtime`

Sync model parameters to real-time application parameters

Syntax

```
syncWithApp(parameter_set,app_name)
```

Description

`syncWithApp(parameter_set,app_name)` synchronizes the parameter name-value pairs and synchronizes the model checksum saved in the parameter set object with the real-time application.

A typical usage for the `syncWithApp` command occurs when you create a new model from an old model by adding or removing several blocks with tunable parameters. You would like to use the parameter set saved from the old model. But, directly loading the old parameter set to the new model generates an error because the number of parameters and model checksum do not match the new model. The `syncWithApp` command adds or removes the unmatched parameters from the parameter set. The command also updates the checksum, which lets you reuse the parameter set saved from the old model.

Examples

Sync Parameter Set with Values from Model

To update the parameter set object with the parameter values from the model, use the `syncWithApp` command.

```
syncWithApp(myParamSet,mdlName);
```

Input Arguments

`parameter_set` — `ParameterSet` object
`ParameterSet` object

The `ParameterSet` object that was created from the real-time application in the `importParamSet` command.

Example: `myParamSet`

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

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

Example: `'slrt_ex_osc'`

Version History

Introduced in R2021a

See Also

`exportParamSet` | `getparam` | `listParamSet` | `loadParamSet` | `saveParamSet` | `ParameterSet` | `Target`

Topics

“Save and Reload Parameters by Using the MATLAB Language”

SystemLog

Get current console log from target computer

Description

A `SystemLog` object represents the console log from the target computer at the time that the object is created by 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, 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 the Simulink Real-Time System Log Viewer tab in the Simulink Real-Time Explorer to view the console log from the 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    Categor
    ____________________    ____________________________________________    ________    _______
```

```
26-Nov-2019 21:27:33    "Target IP address: 192.168.7.5"                    "info"        2
26-Nov-2019 21:28:44    "Loading model slrt_ex_mds_and_tasks"               "info"        0
26-Nov-2019 21:28:44    "Loading model slrt_ex_mds_and_tasks"               "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"
```

Add Custom Messages to System Log

1. To use the custom message functions in an S-function, include the header file. Place an `ifdef` statement around the include.

   ```
   #ifdef SIMULINK_REAL_TIME
   #include "slrt_log.hpp"
   #endif
   ```
2. In an S-function, place an `ifndef` around the function call.

   ```
   #ifdef SIMULINK_REAL_TIME
   slrealtime::log_error("Some custom message.");
   #endif
   ```
3. Call the function for the selected severity level.

   ```
   slrealtime::log_trace("Some custom message.");
   ```

   or

   ```
   slrealtime::log_debug("Some custom message.");
   ```

   or

   ```
   slrealtime::log_info("Some custom message.");
   ```

   or

   ```
   slrealtime::log_warning("Some custom message.");
   ```

   or

   ```
   slrealtime::log_error("Some custom message.");
   ```

   or

   ```
   slrealtime::log_fatal("Some custom message.");
   ```

Version History

Introduced in R2020b

See Also

`slrtLogViewer` | `log_trace` | `log_debug` | `log_info` | `log_warning` | `log_error` | `log_fatal`

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 1-219

`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 1-219

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
getBufferedData	Gets data from the real-time application instrument buffer
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)string
inst.connectScalar(app.Lamp1, "TrafficLight", 'PropertyName', 'Visible', 'Callback', @(t,d)string
inst.connectScalar(app.Lamp2, "TrafficLight", 'PropertyName', 'Visible', 'Callback', @(t,d)string
```

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

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

tg=slrealtime;
tg.addInstrument(inst);

inst.AxesTimeSpan = 10;

inst.AxesTimeSpanOverrun = 'wrap';

inst.AxesTimeSpan = Inf;

tg.removeInstrument(inst);
```

Version History

Introduced in R2020b

See Also

`slrealtime.instrument.LineStyle` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectLine` | `connectScalar` | `delete` | `generateScript` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

Topics

“Instrumentation Apps for Real-Time Applications”

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`

Version History

Introduced in R2020b

See Also

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

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

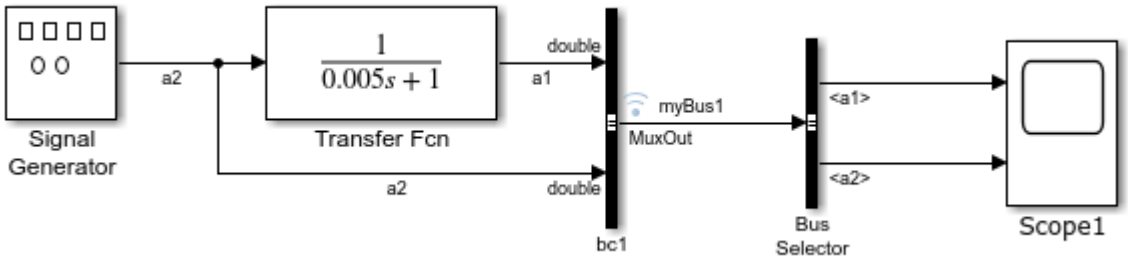
Add a signal for streaming to the real-time instrument object by using the signal name.

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

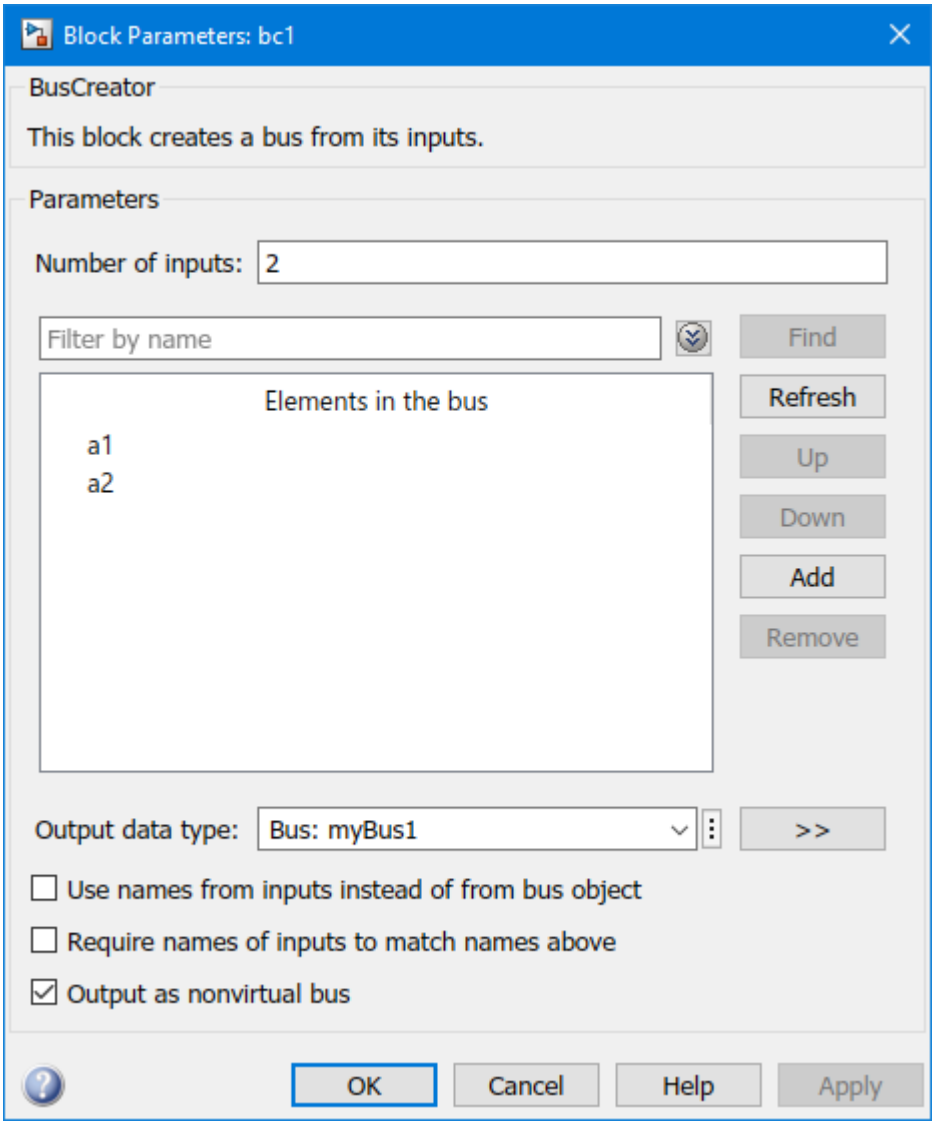
Add Bus Signal by Using BusElement Option

Add bus signals to the real-time instrument object by using the `BusElement` option.

1. In the model `test1`, the block port outputs a bus signal of type `myBus1`, which has bus elements `a1` and `a2`.



2 To view the bus elements, select bus `bc1` and double-click the bus.



3 To instrument these bus elements, use this `addSignal` syntax.

```
% added bus signals to instrument
mldatxfile = 'test1.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
```

```
addSignal(hInst,'test1/bc1',1,'BusElement','a1')
addSignal(hInst,'test1/bc1',1,'BusElement','a2')
```

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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`

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

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. For more information about using the `@my_callback` function handle, see “Listener Callback Syntax”.

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`

Version History

Introduced in R2020b

See Also

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

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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
`slrealtime.instrument.LineStyle` object

Select an `slrealtime.instrument.LineStyle` object that customizes the line appearance. For more information about object, see `slrealtime.instrument.LineStyle`.

Example: `'LineStyle', objectName`

Version History

Introduced in R2020b

R2022b: Select signals by name

In R2022b, you can use a signal name instead of a full block path to create robust binding between a signal and an instrument. This support applies to the `SignalTable` component and functions such as `connectLine` or `connectScalar`.

See Also

`Instrument` | `slrealtime.instrument.LineStyle` | `addInstrumentedSignals` | `addSignal` | `clearScalarAndLineData` | `connectCallback` | `connectScalar` | `delete` | `generateScript` | `getBufferedData` | `getCallbackDataForSignal` | `removeCallback` | `removeSignal` | `validate`

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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`

Version History

Introduced in R2020b

R2022b: Select signals by name

In R2022b, you can use a signal name instead of a full block path to create robust binding between a signal and an instrument. This support applies to the `SignalTable` component and functions such as `connectLine` or `connectScalar`.

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

getBufferedData

Package: `slrealtime`

Gets data from the real-time application instrument buffer

Syntax

```
map_object = getBufferedData(instrument_object)
```

Description

`map_object = getBufferedData(instrument_object)` gets data from buffer on the target computer for real-time application instrument.

In normal (non-buffered) data mode, an instrument for a real-time application can be configured with a callback that gets executed each time data is available. The normal data mode operation of an instrument **pushes** the data to the callback.

In buffered data mode (when the `instrument.BufferedData` flag is enabled), the callback configured for the instrument is not executed when data arrives. Instead, the real-time application stores the data and an app (for example, an instrument panel app) can retrieve the data by using the `getBufferedData` function. The call to this function returns the buffered data, then the function deletes the data from the instrument. The instrument continues to buffer new data until the next call to the `getBufferedData` function. The buffered mode operation of an instrument lets the app **pull** the data from the instrument.

An example situation for buffered data mode is the case when callbacks cannot be executed through the Python-MATLAB bridge. When you create an instrument from Python code, there is no way for the instrument to execute a callback in Python because the Python-MATLAB API does not support callbacks. The buffered data mode lets you get the data from the instrument by requesting it.

Examples

Get Buffered Data for Instrument

Get buffered data for a real-time application instrument by using the `getBufferedData` function. The function gets data that is buffered from the time that you change the real-time application from normal (not buffered) data mode to buffered data mode by enabling the `Instrument.BufferedData` flag. This flag only affects the operation of the `getBufferedData` function.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
addSignal(hInst,'slrt_ex_tank/ControlValue',1);
% . . . change from normal (not buffered) data mode
% . . . change to buffere data mode
Instrument.BufferedData = true;
% . . . inside the my_callback (an slrealtime.instrument callback),
```

```
% . . . you can call getBufferedData
myData = getBufferedData(hInst);
```

Input Arguments

`instrument_object` — Object that represents real-time instrument

object

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

Example: `hInst`

Output Arguments

`map_object` — Object that returns signal data

signal data

The `map_object` returns the buffered data from the `getBufferedData` function call.

Version History

Introduced in R2022b

See Also

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

getCallbackDataForSignal

Package: `slrealtime`

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

Syntax

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

Description

`[time,data] = getCallbackDataForSignal(instrument_object,event_data, 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,event_data, 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. Because the `getCallbackDataForSignal` function takes event data as an input argument, call this function from inside an `slrealtime.instrument` callback function.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst = slrealtime.Instrument(mldatxfile);
connectCallback(hInst,@my_callback);
addSignal(hInst,'slrt_ex_tank/ControlValue',1);
% . . . inside the my_callback (an slrealtime.instrument callback),
% . . . you can call getCallbackDataForSignal
[cv_time,cv_data] = getCallbackDataForSignal(hInst,hEvnt,'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. Because the `getCallbackDataForSignal` function takes event data as an input argument, call this function from inside an `slrealtime.instrument` callback function.

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

```
addSignal(hInst,'ControlValue');
% . . . inside the my_callback (an slrealtime.instrument callback),
% . . . you can call getCallbackDataForSignal
[cv_time,cv_data] = getCallbackDataForSignal(hInst,hEvnt,'ControlValue');
```

Input Arguments

`instrument_object` — Object that represents real-time instrument

object

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

Example: `hInst`

`event_data` — Event that triggers the callback operation

object

The `eventData` object identifies the event that triggers callback operation.

Example: `hEvnt`

`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.

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

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`

Version History

Introduced in R2020b

See Also

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

slrealtime.instrument.LineStyle

Create real-time instrument LineStyle object

Description

An `slrealtime.instrument.LineStyle` object defines line style properties for a `connectLine` function.

Creation

`lineStyle_object = slrealtime.instrument.LineStyle()` creates an empty instrument `LineStyle` object for setting line style properties.

Example: “Configure Line Style Properties for connectLine” on page 1-249

Properties

`Color` — Line color
`[-1 -1 -1]`(auto select) (default) | `'red'` | `'r'` | `'green'` | `'g'` | `'blue'` | `'b'` | `'cyan'` | `'c'` | `'magenta'` | `'m'` | `'yellow'` | `'y'` | `'black'` | `'k'` | `'white'` | `'w'`

Select the line color.

Example: `myLineStyle.Color = 'black';`

`Marker` — Marker character
`'none'` (default) | `+` | `'o'` | `'*'` | `'.'` | `'x'` | `'square'` | `'diamond'` | `'v'` | `'^'` | `'>'` | `'<'` | `'pentagram'` | `'hexagram'`

Select the marker character in the line.

Example: `myLineStyle.Marker = '*';`

`MarkerSize` — Marker character size
`6` (default) | positive scalar value

Select the marker character size in the line.

Example: `myLineStyle.MarkerSize = 4;`

`Style` — Line style character
`'-'` (default) | `'--'` | `':'` | `'-.'`

Select the line style character.

Example: `myLineStyle.Style = '--';`

`Width` — Line width
`0.5` (default) | positive scalar value

Select the line width

Example: `myLineStyle.Width = 1;`

Examples

Configure Line Style Properties for connectLine

- You can configure style properties in a `LineStyle` object and apply the style in the `connectLine` function.

```
mldatxfile = 'slrt_ex_tank.mldatx';
hInst  = slrealtime.Instrument(mldatxfile);
myLineStyle = slrealtime.instrument.LineStyle();
myLineStyle.Marker = '*';
myLineStyle.MarkerSize = 4;
myLineStyle.Color = 'black';
connectLine(hInst,myAxis,'ControlValueOut', ...
    'LineStyle', myLineStyle);
```

Version History

Introduced in R2022b

See Also

`Instrument` | `connectLine`

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 Profiler and the SLRT Overload Options block use different mechanisms to measure TET and do not generate identical TET values.

Creation

`getProfilerData`

Object Functions

plot	Generate execution profiler plot
report	Generate profiler report

Examples

Run Profiler and Explicitly Display Profiler Data

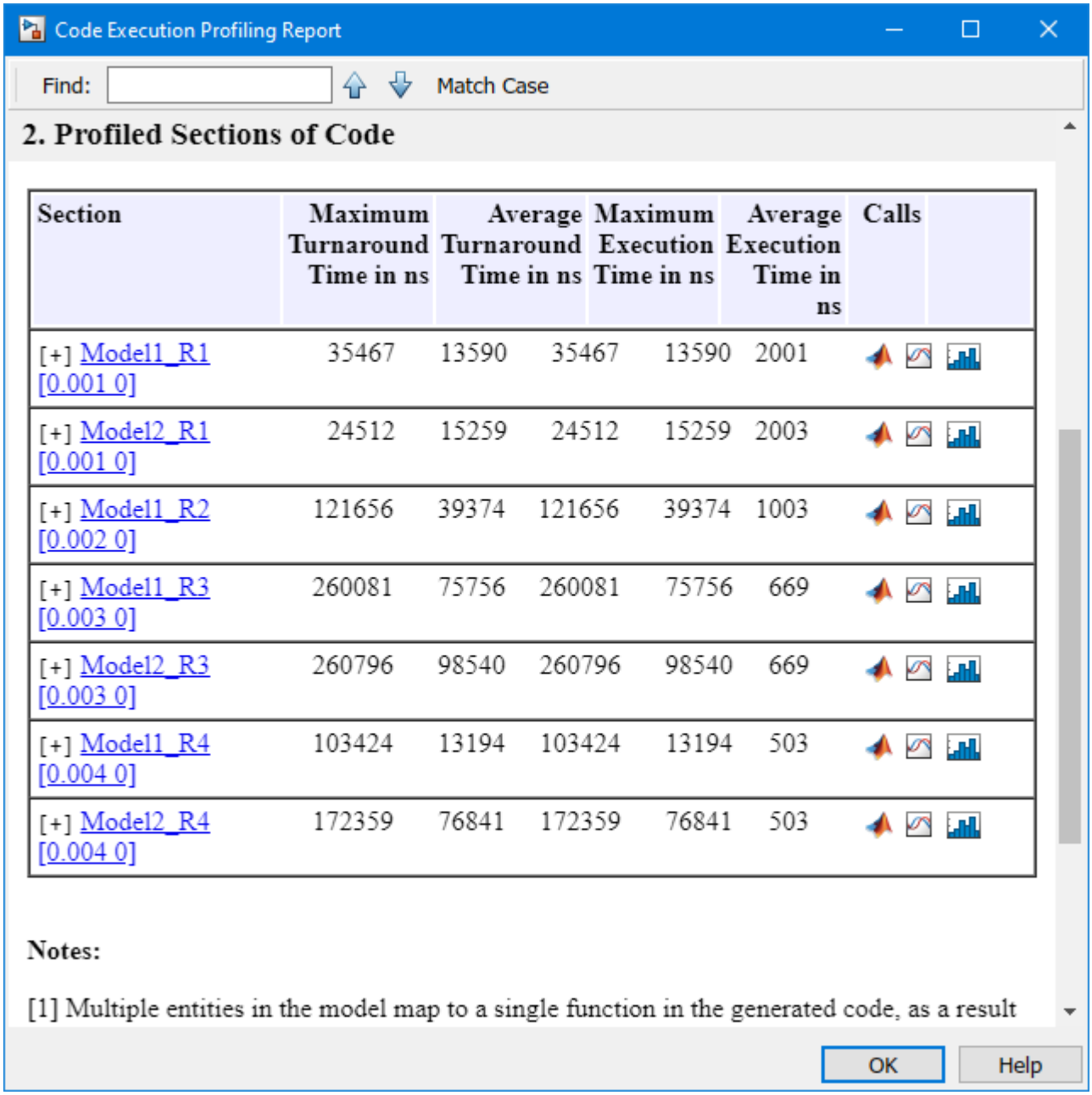
Load the application. Start the profiler. Start the application. Stop the profiler. Retrieve profile execution data. Call `report` and `plot` on the data.

```
tg = slrealtime('TargetPC1');
slbuild('slrt_ex_mds_and_tasks');
load(tg,'slrt_ex_mds_and_tasks');
startProfiler(tg);
start(tg);
stopProfiler(tg);
stop(tg);
profiler_object = getProfilerData(tg);
```

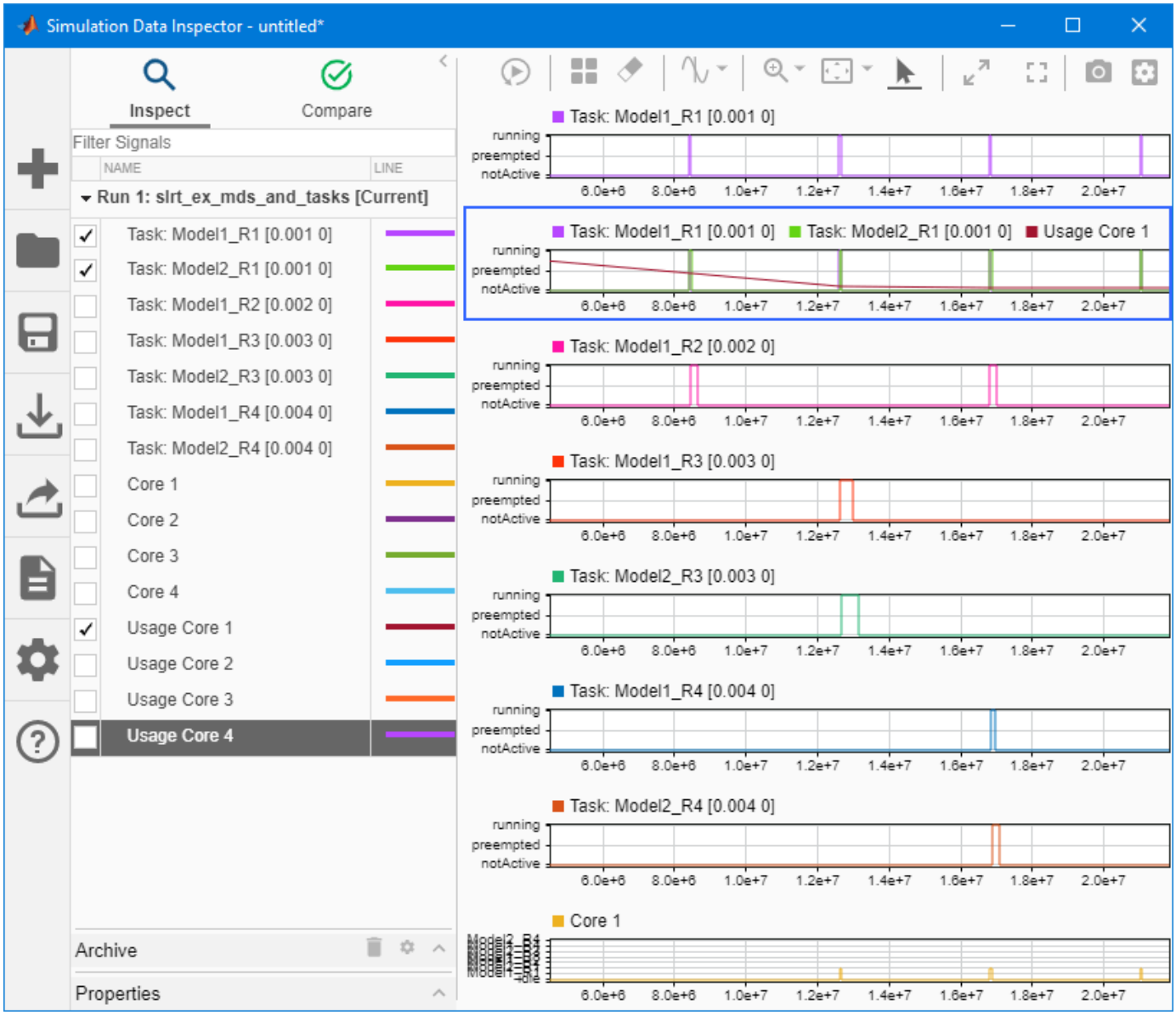
```
Processing data on target computer, please wait ...
Transferring data from target computer to host computer, please wait ...
```

```
Processing data on host computer, please wait ...

Code execution profiling data for model slrt_ex_mds_and_tasks.
```

```
4  report(profiler_object);
```




```
5  plot(profiler_object);
```



Version History

Introduced in R2020b

See Also

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

Topics

“Execution Profiling for Real-Time Applications”

plot

Package: `slrealtime`

Generate execution profiler plot

Syntax

```
plot(profiler_object)
```

Description

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

The Execution Profiler and the SLRT Overload Options block use different mechanisms to measure TET and do not generate identical TET values.

Examples

Run Profiler and Plot Profiler Data

The real-time application is already loaded. Start the profiler. Start the application.

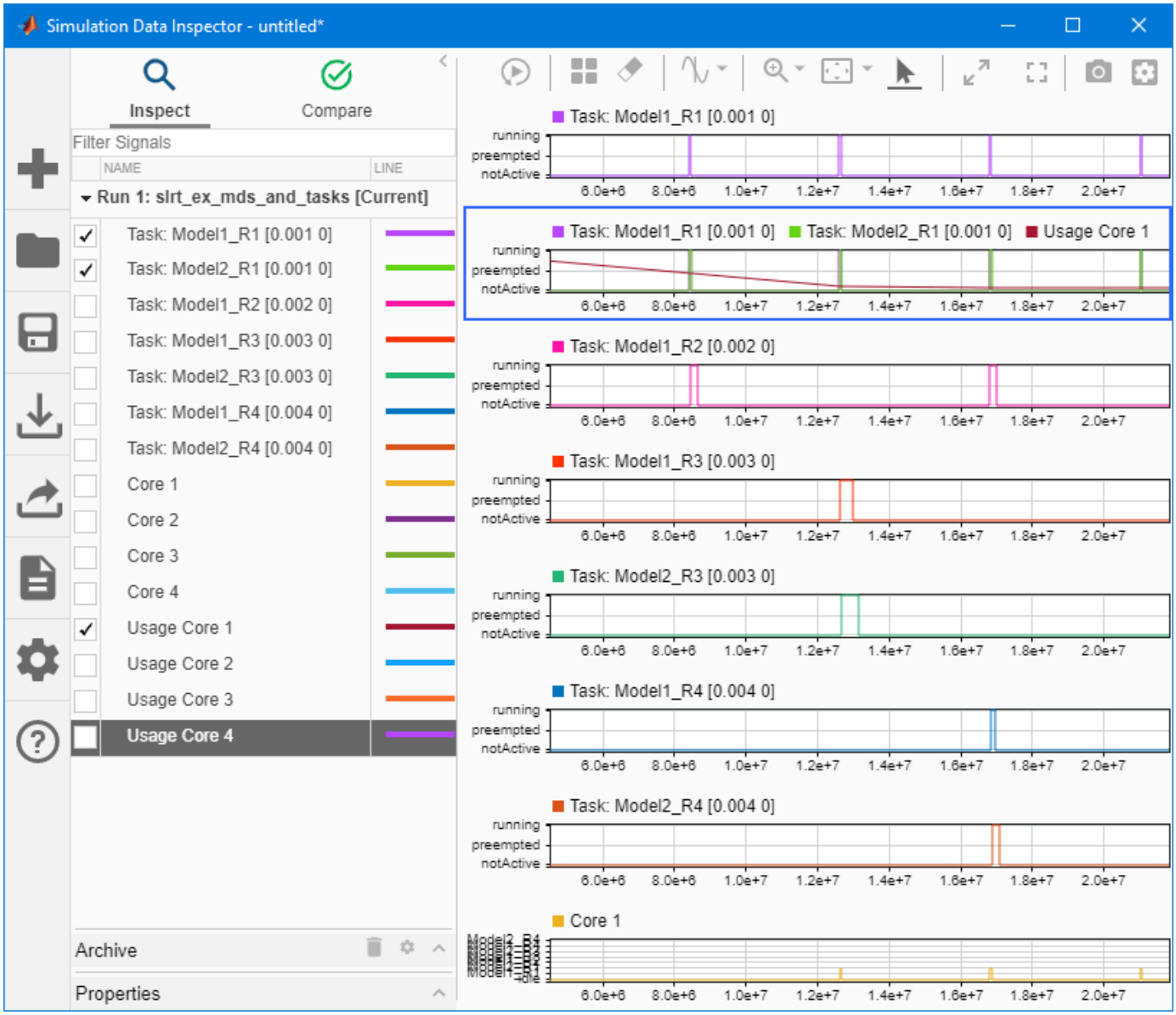
1.
```
tg = slrealtime('TargetPC1');
startProfiler(tg);
start(tg);
```
2. Stop the profiler. Stop the application.

```
stopProfiler(tg);
stop(tg);
```
3. Retrieve profiler data.

```
profiler_object = getProfilerData(tg);
```

```
Processing data, please wait ...
```
4. 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.

Version History

Introduced in R2020b

See Also

`report` | `ProfilerData` | `getProfilerData`

Topics

“Execution Profiling for Real-Time Applications”

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.

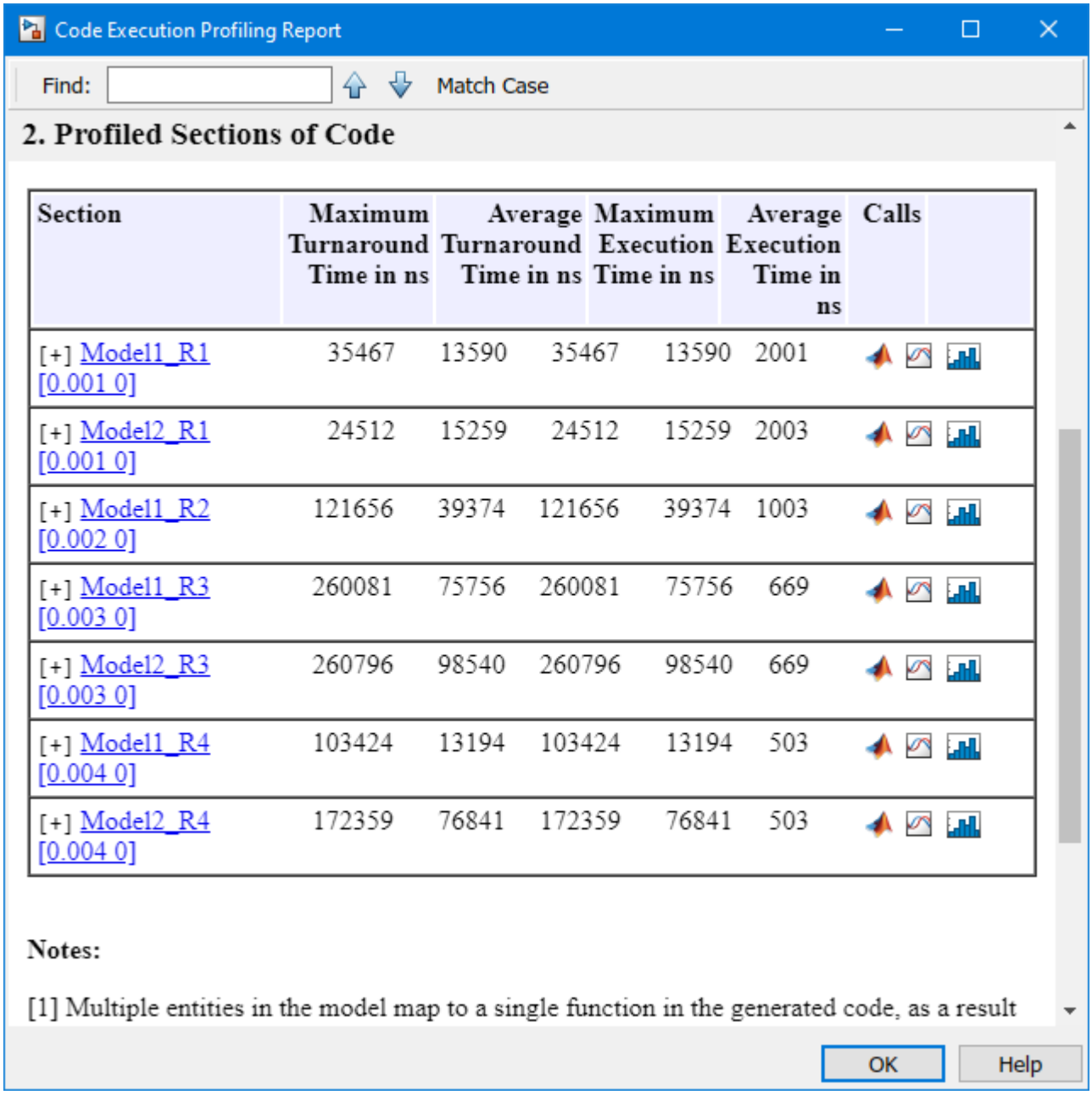
1.
```
tg = slrealtime('TargetPC1');
startProfiler(tg);
start(tg);
```
2. Stop the profiler. Stop the application.

```
stopProfiler(tg);
stop(tg);
```
3. Retrieves profiler data.

```
profiler_object = getProfilerData(tg);
```

```
Processing data, please wait ...
```
4. Call the `report` function on the results data.

```
report(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.

Version History

Introduced in R2020b

See Also

`plot` | `ProfilerData` | `getProfilerData`

Topics

“Execution Profiling for Real-Time Applications”

Menu

Package: `slrealtime`

Create menu of commands for instrument panel UI

Syntax

```
hMenu = slrealtime.ui.container.Menu(hFigure)
```

Description

`hMenu = slrealtime.ui.container.Menu(hFigure)` adds a menu to an existing menu or to a figure for an instrument panel `uifigure` figure. The menu commands include:

- Select a target computer
- Connect or disconnect a target computer
- Load a real-time application on a target computer
- Start or stop a real-time application running on a target computer
- Update software on a target computer

For information about the `slrealtime.ui.container.Menu` component `TargetSelector` property, see slrealtime.ui Properties.

Examples

Create Menu

Creates a menu of commands for the instrument panel.

```
% Create figure
hFig = uifigure();
% Create hMenu
hUIMenu = uimenu(hFig);
% Create Menu component
hSlrtTMenu = slrealtime.ui.container.Menu(hFig);
hSlrtUIMenu = slrealtime.ui.container.Menu(hUIMenu, 'Name', 'mymenu');
```

Create Menu by Using App Generator

If you select **Options** > **Menu** in the Simulink Real-Time App Generator, the generator creates a menu that provides target computer controls. In the `startupFcn` of the generated instrument panel app, the Code View shows how this menu sets property values for this menu.

```
menu = slrealtime.ui.container.Menu(app.UIFigure);
targetSelector = menu.TargetSelector;
menu.SkipInstall = 0;
menu.AsyncLoad = 0;
```

```
menu.ReloadOnStop = 1;
menu.AutoImportFileLog = 1;
menu.ExportToBaseWorkspace = 1;
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Name', 'mymenu'`

`Name` — Set name for custom menu

string

The `Name` name-value argument selects a name for the menu that you create.

Example: `'Name', 'mymenu'`

Data Types: `string`

Output Arguments

`hMenu` — Handle to menu component

component handle

The `hMenu` argument is the handle to the menu component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

ConnectButton

Package: `slrealtime`

Create connect target computer button for instrument panel UI

Syntax

```
hConnectButton = slrealtime.ui.control.ConnectButton(hFigure)
```

Description

`hConnectButton = slrealtime.ui.control.ConnectButton(hFigure)` creates a target computer connect-disconnect button for an instrument panel `uifigure` figure. This single button has two states.

- For the connected state, the button indicates that the development computer is connected to target computer. Clicking the button disconnects the development computer from the target computer.
- For the disconnected state, the button indicates that the development computer is disconnected from target computer. Clicking the button connects the development computer to the target computer.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Connect-Disconnect Button

Create a target computer connect-disconnect button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create connect button component
hConnected = slrealtime.ui.control.ConnectButton(hFig);
% Change position of the component
hConnected.Position = [0 0 200 200];
% Associate with a target object
hConnected.TargetSource = 'TargetPC1';
% Customize
hConnected.ConnectedIcon = fullfile(pwd, 'myConnectedIcon.png');
hConnected.DisconnectedIcon = fullfile(pwd, 'myDisconnectedIcon.png');
hConnected.ConnectedText = 'Push to disconnect';
hConnected.DisconnectedText = 'Push to connect';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hConnectButton` — Handle to connect button component
component handle

The `hConnectButton` argument is the handle to the connect button component that you create.

Version History

Introduced in R2021b

See Also

`LoadButton` | `InstrumentManager` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics
“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

LoadButton

Package: `slrealtime`

Create load real-time application button for instrument panel UI

Syntax

```
hLoadButton = slrealtime.ui.control.LoadButton(hFigure)
```

Description

`hLoadButton = slrealtime.ui.control.LoadButton(hFigure)` creates a target computer load button for an instrument panel `uifigure` figure. This button loads a real-time application onto the target computer and optionally displays the currently loaded application. You can disable this display by using a button property. The button has properties that enable you to set a loading option for skip install and asynchronous load.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Load Button

Create a real-time application load button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create load application component
hLoaded = slrealtime.ui.control.LoadButton(hFig);
% Change position of the component
hLoaded.Position = [0 0 200 200];
% Associate with a Target Select component
hLoaded.TargetSource = 'TargetPC1';
% Customize
hLoaded.LoadIcon = fullfile(pwd, 'myLoadIcon.png');
hLoaded.LoadText = 'push to load';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hLoadButton` — Handle to load button component
component handle

The `hLoadButton` argument is the handle to the load button component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics
“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

ParameterTable

Package: `slrealtime`

Create parameter table for instrument panel UI

Syntax

```
hParamTable = slrealtime.ui.control.ParameterTable(hFigure)
```

Description

`hParamTable = slrealtime.ui.control.ParameterTable(hFigure)` creates an editable parameter table display for an instrument panel `uifigure` figure. The display shows the tunable parameters that have been selected for streaming in the real-time application. When the instrument panel app is running, the `ParameterTable` component provides a right-click menu that lets you select parameters to add to or remove from the display.

When the ECU page and XCP page selections do not match, the mismatch disables the App Designer `ParameterTable` component and `ParameterTuner` component. You can enable operation of these components by coordinating ECU page and XCP page selection in the real-time application. Use the `getECUPage`, `setECUPage`, `getXCPPage`, and `setXCPPage` functions. Or, use the explorer **Enable Parameter Table** button. This button is context sensitive and appears when explorer detects a page selection mismatch.

For information about display properties, see slrealtime.ui Properties.

Examples

Create Parameter Table

Create a parameter table and adjust the position of the table. When using workspace variables in the `Parameters` property, the `BlockPath` is empty for the workspace variable, and the `ParameterName` value is the name of the workspace variable.

```
hFig = uifigure();
% Create parameter table component
hPTable = slrealtime.ui.control.ParameterTable(hFig);
hPTable.Parameters = struct( ...
  'BlockPath', {'testmodel/Constant1', '', ...
         'testmodel/Constant5', '', ...
         'testmodel/str1', 'testmodel/str2', ...
         'testmodel/multi-line block name', ...
         'testmodel/Constant6'}, ...
  'ParameterName', {'Value', 'model_wksp_var', ...
           'Value', 'base_wksp_var', 'String', ...
           'String', 'Value', 'Value'});
% Change position of the component
hPTable.Position = [0 0 200 200];
% Customize
```

```
hPTable.TableBackgroundColor = [0 0 1]; % 'blue'
hPTable.TableForegroundColor = [1 1 0]; % 'yellow'
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hParamTable` — Handle to parameter table component

component handle

The `hParamTable` argument is the handle to the parameter table component that you create.

Version History

Introduced in R2021b

R2022b: Added context menu for selecting parameters at run time

Added a right-click menu to the `ParameterTable` component that lets you select parameters to add to or remove from the display while the instrument panel app is running.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTuner` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

RebootButton

Package: `slrealtime`

Create reboot target computer button for instrument panel UI

Syntax

```
hRebootButton = slrealtime.ui.control.RebootButton(hFigure)
```

Description

`hRebootButton = slrealtime.ui.control.RebootButton(hFigure)` creates a target computer reboot button for an instrument panel `uifigure` figure. This button reboots the target computer. You can disable this display by using a button property. The button has a property that enables you to pause for completion of the reboot operation.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Reboot Button

Create a real-time application reboot button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create load application component
hRebooted = slrealtime.ui.control.RebootButton(hFig);
% Change position of the component
hLoaded.Position = [0 0 200 200];
% Pause for completion of reboot
hRebooted.WaitForReboot = 1;
% Customize
hRebooted.RebootIcon = fullfile(pwd, 'myRebootIcon.png');
hRebooted.RebootText = 'push to reboot';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hRebootButton` — Handle to reboot button component
component handle

The `hRebootButton` argument is the handle to the reboot button component that you create.

Version History

Introduced in R2022b

See Also

`ConnectButton` | `InstrumentManager` | `Menu` | `ParameterTable` | `ParameterTuner` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics
“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

RecordButton

Package: `slrealtime`

Create record (log and stream signals) real-time application button for instrument panel UI

Syntax

```
hRecordButton = slrealtime.ui.control.RecordButton(hFigure)
```

Description

`hRecordButton = slrealtime.ui.control.RecordButton(hFigure)` creates a target computer record button for an instrument panel `uifigure` figure. This button starts and stops logging and streaming signals from a real-time application on the target computer. You can disable this display by using a button property.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Record Button

Create a real-time application start/stop recording button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create record application component
hRecord = slrealtime.ui.control.RecordButton(hFig);
% Change position of the component
hRecord.Position = [0 0 200 200];
% Customize
hRecord.StartRecordingIcon = fullfile(pwd, 'myStartRecordingIcon.png');
hRecord.StartRecordingText = 'push to start recording';
hRecord.StopRecordingIcon = fullfile(pwd, 'myStopRecordingIcon.png');
hRecord.StopRecordingText = 'push to stop recording';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hRecordButton` — Handle to record button component
component handle

The `hRecordButton` argument is the handle to the start/stop recording button component that you create.

Version History

Introduced in R2022b

See Also

`ConnectButton` | `InstrumentManager` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics
“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

SignalTable

Package: `slrealtime`

Create signal table for instrument panel UI

Syntax

```
hSigTable = slrealtime.ui.control.SignalTable(hFigure)
```

Description

`hSigTable = slrealtime.ui.control.SignalTable(hFigure)` creates an editable signal table display for an instrument panel `uifigure` figure. Only the **Enabled** column is editable to enable or disable a signal from streaming its value to the table. The display shows the signals that you have selected for streaming in the real-time application. When the instrument panel app is running, the `SignalTable` component provides a right-click menu that lets you select signals to add to or remove from the display.

For information about display properties, see slrealtime.ui Properties.

Examples

Create Signal Table

Create a signal table and adjust the position of the table.

```
% Create figure
hFig = uifigure();
% Create signal table component
hSTable = slrealtime.ui.control.SignalTable(hFig);
hSTable.Signals = struct( ...
  'BlockPath', {'testmodel/Constant1', 'testmodel/Constant2', ...
                'testmodel/Sine Wave', 'testmodel/str2', ...
                'testmodel/Switch1'}, ...
  'PortIndex', {1, 1, 1, 1, 1}, . . .
  'Decimation', {1, 2, 5, 1, 1}, . . .
  'ArrayIndex', {[], [], 2, [2 2], 3}, . . .
  'BusElement', {'', 'b', '', 'z', 'a'}, . . .
  'Callback', {[], cb, [], [], []});
% Change position of the component
hSTable.Position = [0 0 200 200];
% Customize
hSTable.FontWeight = 'bold';
hSTable.FontAngle = 'italic';
```

Create Signal Table by Using Signal Names

Create a signal table and adjust the position of the table. When using signal names in the `Signals` property, the `BlockPath` selects the signal name and the `PortIndex` value is `-1`.

```
% Create figure
hFig = uifigure();
% Create signal table component
hSTable = slrealtime.ui.control.SignalTable(hFig);
hSTable.Signals = struct( ...
  'BlockPath', {'testmodel/Constant1', 'testmodel/Constant2', ...
                'Sine', 'String', 'testmodel/Switch1'}, ...
  'PortIndex', {1, 1, -1, -1, 1});
  'Decimation', {1, 2, 5, 1, 1}, . . .
  'ArrayIndex', {[], [], 2, [2 2], 3}, . . .
  'BusElement', {'', 'b', '', 'z', 'a'}, . . .
  'Callback', {[], cb, [], [], []});
% Change position of the component
hSTable.Position = [0 0 200 200];
% Customize
hSTable.FontWeight = 'bold';
hSTable.FontAngle = 'italic';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hSigTable` — Handle to signal table component

component handle

The `hSignTable` argument is the handle to the signal table component that you create.

Version History

Introduced in R2021b

R2023a: Added optional properties for Decimation, BusElement, ArrayIndex, and Callback

The `SignalTable` provides optional properties that let you configure `Decimation`, `BusElement`, `ArrayIndex`, and `Callback` for each signal added to the signal table. For more information, see the Signals property.

R2022b: Added context menu for selecting signals at run time

Added a right-click menu to the `SignalTable` component that lets you select signals to add to or remove from the display while the instrument panel app is running.

R2022b: Select signals by name

In R2022b, you can use a signal name instead of a full block path to create robust binding between a signal and an instrument. This support applies to the `SignalTable` component and functions such as `connectLine` or `connectScalar`.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

SimulationTimeEditField

Package: `slrealtime`

Create simulation time edit field component for instrument panel UI

Syntax

```
hSimulationTime = slrealtime.ui.control.SimulationTimeEditField(hFigure)
```

Description

`hSimulationTime = slrealtime.ui.control.SimulationTimeEditField(hFigure)` creates a real-time application simulation time edit field for an instrument panel `uifigure` figure. This field displays the simulation time of the current application.

For information about field properties, see slrealtime.ui Properties.

Examples

Create Application Simulation Time Field

Create a real-time application simulation time edit field and adjust the position of the field.

```
% Create figure
hFig = uifigure();
% Create simulation time component
hSimTime = slrealtime.ui.control.SimulationTimeEditField(hFig);
% Change position of the component
hSimTime.Position = [0 0 200 200];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hSimulationTime` — Handle to simulation time field component

component handle

The `hSimulationTime` argument is the handle to the simulation time field component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

StartStopButton

Package: `slrealtime`

Create start-stop real-time application button for instrument panel UI

Syntax

```
hStartStopButton = slrealtime.ui.control.StartStopButton(hFigure)
```

Description

`hStartStopButton = slrealtime.ui.control.StartStopButton(hFigure)` creates a target computer start-stop button for an instrument panel `uifigure` figure. This button starts or stops a real-time application running on the target computer and displays the state of the current application.

- When the button displays the **Start** icon and text, the target computer is not running an application. Clicking the button component starts the application.
- When the button displays the **Stop** icon and text, the target computer is running an application. Clicking the button component stops the application.

The button provides an option to `Reload On Stop`.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Start-Stop Button

Create a real-time application start-stop button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create start/stop component
hStarted = slrealtime.ui.control.StartStopButton(hFig);
% Change position of the component
hStarted.Position = [0 0 200 200];
% Associate with default target
hStarted.TargetSource = [];
% Configure start options
hStarted.ReloadOnStop = 1;
hStarted.AutoImportFileLog = 1;
hStarted.ExportToBaseWorkspace = 1;
% Customize
hStarted.StartIcon = fullfile(pwd, 'start.png');
hStarted.StartText = 'START ME';
```

```
hStarted.StartIcon = fullfile(pwd, 'stop.png');
hStarted.StartText = 'STOP ME';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hStartStopButton` — Handle to start-stop button component

component handle

The `hStartStopButton` argument is the handle to the start-stop button component that you create.

Version History

Introduced in R2021b

R2022b: Added properties for application start and stop options

Added `ReloadOnStop` property, `AutoImportFileLog` property, and `ExportToBaseWorkspace` property to .`StartStopButton` component. For information about button properties, see slrealtime.ui Properties.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `SignalTable` | `SimulationTimeEditField` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

StatusBar

Package: `slrealtime`

Create status bar for instrument panel UI

Syntax

```
hSBar = slrealtime.ui.control.StatusBar(hFigure)
```

Description

`hSBar = slrealtime.ui.control.StatusBar(hFigure)` creates a status bar display for an instrument panel `uifigure` figure. This bar provides information that is similar to the status bar display in Simulink Real-Time Explorer. For example, when recording stops, the `StatusBar` displays a message similar to the message that appears in explorer.

For information about display properties, see slrealtime.ui Properties.

Examples

Create Status Bar Display

Create a status bar display and adjust the position of the display.

```
% Create figure
hFig = uifigure();
% Create simulation time component
hStatus = slrealtime.ui.control.StatusBar(hFig);
% Change position of the component
hStatus.Position = [0 0 200 200];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hSBar` — Handle to status bar display component

component handle

The `hSBar` argument is the handle to the status bar display component that you create.

Version History

Introduced in R2021b

R2023a: Added recording stops message

The `StatusBar` displays a message when recording stops, similar to the message that is displayed by Simulink Real-Time Explorer.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

StopTimeEditField

Package: `slrealtime`

Create stop time edit field component for instrument panel UI

Syntax

```
hStopTime = slrealtime.ui.control.StopTimeEditField(hFigure)
```

Description

`hStopTime = slrealtime.ui.control.StopTimeEditField(hFigure)` creates a real-time application stop time edit field for an instrument panel `uifigure` figure. This editable field displays the stop time of the current application.

For information about field properties, see slrealtime.ui Properties.

Examples

Create Application Stop Time Field

Create a real-time application stop time edit field and adjust the position of the field.

```
% Create figure
hFig = uifigure();
% Create stop time component
hSTime = slrealtime.ui.control.StopTimeEditField(hFig);
% Change position of the component
hSTime.Position = [0 0 200 200];
% Customize
hSTime.BackgroundColor = 'red';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hStopTime` — Handle to stop time field component

component handle

The `hStopTime` argument is the handle to the stop time field component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

SystemLog

Package: `slrealtime`

Create system log component for instrument panel UI

Syntax

```
hSysLog = slrealtime.ui.control.SystemLog(hFigure)
```

Description

`hSysLog = slrealtime.ui.control.SystemLog(hFigure)` creates a target computer system log display for an instrument panel `uifigure` figure. The display has a property to set time stamp option to `Include Time Stamps`.

For information about display properties, see slrealtime.ui Properties.

Examples

Create System Log Display

Create a target computer system log display and adjust the position of the display.

```
% Create figure
hFig = uifigure();
% Create simulation time component
hLog = slrealtime.ui.control.SystemLog(hFig);
% Change position of the component
hLog.Position = [0 0 200 200];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hSysLog` — Handle to system log component

component handle

The `hSysLog` argument is the handle to the system log component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

TargetSelector

Package: `slrealtime`

Create target computer selector component for instrument panel UI

Syntax

```
hTargetSelector = slrealtime.ui.control.TargetSelector(hFigure)
```

Description

`hTargetSelector = slrealtime.ui.control.TargetSelector(hFigure)` creates a target computer selector component for an instrument panel `uifigure` figure. Simulink Real-Time refers to target computers by mapping unique names to IP addresses. Multiple mappings are allowed.

You can access each target computer by using its name. One mapping is the default and is used when no target computer name is supplied. The target selector component displays the list of all named target computers currently defined on the development computer.

The target computer selector list entry `Simulink Normal Mode` is not a customer-defined target. This selection appears at the end of the list to support Simulink normal mode simulation of a model. When you select `Simulink Normal Mode` in the target selector, the load application button changes to a load model button. Use this button to select the model to interface with the control panel app.

The drop-down list is editable. You can enter a valid IP address and create a temporary target computer mapping that is removed when the target computer selector component is destroyed.

It is possible to have different components in the same instrument panel synchronize with different target computers. By selecting a target computer for a component by using the `TargetSelector`, instrument panel components with their `TargetSource` property set to the `TargetSelector` synchronize and are updated to the current state of the target computer selected by the `TargetSelector`. For more information, see the `TargetSource` property description in slrealtime.ui Properties.

Examples

Create Target Computer Selector

Create a target selector component and adjust the position of the component.

```
% Create figure
hFig = uifigure();
% Create target selector component
hTgSelect = slrealtime.ui.control.TargetSelector(hFig);
```

```
% Change position of the component
hTgSelect.Position = [100 100 200 30];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hTargetSelector` — Handle to target selector component

component handle

The `hTargetSelector` argument is the handle to the target selector component that you create.

Version History

Introduced in R2021b

R2022b: MATLAB Compiler Compatibility

For compatibility with the MATLAB Compiler, if no target computer is set in the instrument panel app, the initial value for the `TargetSelector` control is `Enter_IP_Address_Here`.

R2022a: Simulink Normal Mode Option

The `TargetSelector` control for instrument panel apps provides a **Simulink Normal Mode** selection. This selection lets you interface the instrument panel with a normal mode simulation of a model.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

TETMonitor

Package: `slrealtime`

Create TET monitor component for instrument panel UI

Syntax

```
hMonitor = slrealtime.ui.control.TETMonitor(hFigure)
```

Description

`hMonitor = slrealtime.ui.control.TETMonitor(hFigure)` creates a task execution time (TET) monitor display for an instrument panel `uifigure` figure. The display shows the amount of time a task takes to run one step in the real-time application.

For information about display properties, see slrealtime.ui Properties.

Examples

Create TET Monitor Display

Create a TET monitor display and adjust the position of the display.

```
% Create figure
hFig = uifigure();
% Create TET task component
hTET = slrealtime.ui.control.TETMonitor(hFig);
% Change position of the component
hTET.Position = [0 0 200 200];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hMonitor` — Handle to TET monitor component

component handle

The `hMonitor` argument is the handle to the TET monitor display component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

UpdateButton

Package: `slrealtime`

Create update target computer software button for instrument panel UI

Syntax

```
hUpdateButton = slrealtime.ui.control.UpdateButton(hFigure)
```

Description

`hUpdateButton = slrealtime.ui.control.UpdateButton(hFigure)` creates a target computer software update button for an instrument panel `uifigure` figure. Clicking this button updates the system software on the target computer.

For information about button properties, see slrealtime.ui Properties.

Examples

Create Target Computer Software Update Button

Create a software update button and adjust the position of the button.

```
% Create figure
hFig = uifigure();
% Create connect button component
hUpdated = slrealtime.ui.control.UpdateButton(hFig);
% Change position of the component
hUpdated.Position = [0 0 200 200];
% Associate with a target object
hUpdated.TargetSource = 'TargetPC1';
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hUpdateButton` — Handle to software update button component

component handle

The `hUpdateButton` argument is the handle to the target computer software update button component that you create.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

InstrumentManager

Package: `slrealtime`

Create instrument manager component for App Designer components in instrument panel UI

Syntax

```
hInstManager = slrealtime.ui.tool.InstrumentManager(hFigure)
```

Description

`hInstManager = slrealtime.ui.tool.InstrumentManager(hFigure)` creates an instrument manager component that manages instruments for App Designer components, which you add to an instrument panel `uifigure` figure.

The instrument manager component manages a collection of `Instrument` objects that are used by the instrument panel and provides the features of Simulink Real-Time app components. When an instrument panel switches target computers, the managed instruments are removed from the previous target computer and added to the new target computer.

For information about instrument manager properties, see slrealtime.ui Properties.

Examples

Create Instrument Manager and Add Instruments

Create an instrument manager component and add instruments to it for App Designer components.

```
% Create figure
hFig = uifigure();
hAxes = uiaxes(hFig);
hGauge = uigauge(hFig);
% Create some slrealtime.Instruments
hInstAxes = slrealtime.Instrument();
hInstAxes.connectLine(hAxes, 'SineWave');
hInstGauge = slrealtime.Instrument();
hInstGauge.connectScalar(hGauge, 'BlockPath', 1);
% Create InstrumentManager object and add instruments
hInst = slrealtime.ui.tool.InstrumentManager(hFig);
hInst.Instruments = [hInstAxes hInstGauge];
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hInstManager` — Handle to instrument manager component

component handle

The `hInstManager` argument is the handle to the instrument manager component that manages instruments for the App Designer components.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

ParameterTuner

Package: `slrealtime`

Create parameter tuner component for App Designer component in instrument panel UI

Syntax

```
hPTuner = slrealtime.ui.tool.ParameterTuner(hFigure)
```

Description

`hPTuner = slrealtime.ui.tool.ParameterTuner(hFigure)` creates a parameter tuner component for an App Designer component on an instrument panel `uifigure` figure. After connecting the parameter tuner to an App Designer component (such as a knob), the App Designer component gets or sets data in the real-time application in the same manner as Simulink Real-Time components available in the App Designer. The parameter tuner component pulls the current parameter value from the target when:

- The instrument panel first starts and an application is loaded on the selected target computer.
- The target selected by the instrument panel changes and an application is loaded on the selected target computer.
- An application is loaded on the selected target computer.
- The parameter is changed from an external source, such as the command line, Explorer, other instrument panels on the selected target computer.

If the current value of the parameter is changed from an external source and is invalid for the Parameter Tuning component (for example, it is out of range of the knob), the Parameter Tuning component displays a warning icon covering the component. Changing the parameter from an external source to some valid value for the component removes the warning icon.

When the ECU page and XCP page selections do not match, the mismatch disables the App Designer `ParameterTable` component and `ParameterTuner` component. You can enable operation of these components by coordinating ECU page and XCP page selection in the real-time application. Use the `getECUPage`, `setECUPage`, `getXCPPage`, and `setXCPPage` functions. Or, use the explorer **Enable Parameter Table** button. This button is context sensitive and appears when explorer detects a page selection mismatch.

For information about parameter tuner properties, see slrealtime.ui Properties.

Examples

Connect App Designer Component for Parameter Tuning

Create an App Designer component and connect it to a block in a real-time application for parameter tuning. Changing the value of the component pushes the value to the real-time application on the target computer.

```
% Create figure
hFig = uifigure();
```

```
% Create a hKnob
hKnob = uiknob(hFig);
% Create Parameter Tuning object
hParamTuner = slrealtime.ui.tool.ParameterTuner(hFig);
hParamTuner.Component = hKnob;
hParamTuner.BlockPath = 'testmodel/Constant6';
hParamTuner.ParameterName = 'Value';
```

Connect App Designer Component to Workspace Variable for Parameter Tuning

In “Connect App Designer Component for Parameter Tuning” on page 1-292, the code shows how to connect the `ParameterTuner` component to a value on a block. Instead, you can use slightly different code to connect the `ParameterTuner` component to a variable or parameter in the workspace. In the code for the component, the `BlockPath` is empty, and the `ParameterName` is the parameter name instead of the property of the block to tune. The syntax for the code could be:

```
% Create Parameter Tuning object
hParamTuner = slrealtime.ui.tool.ParameterTuner(hFig);
hParamTuner.Component = hKnob;
hParamTuner.BlockPath = '';
hParamTuner.ParameterName = 'myParameter';
```

Change Component Value Programmatically

The `changeComponentValue` function enables you to change the value of the parameter that is connected to the `ParameterTuner` component. Using this function to change the value of the component pushes the value to the real-time application on the target computer.

```
% Create figure
hFig = uifigure();
% Create a hKnob
hKnob = uiknob(hFig);
% Create Parameter Tuning object
hParamTuner = slrealtime.ui.tool.ParameterTuner(hFig);
hParamTuner.Component = hKnob;
hParamTuner.BlockPath = 'slrt_ex_osc/Signal Generator';
hParamTuner.ParameterName = 'Amplitude';
changeComponentValue(hParamTuner,2)
getparam(tg, 'slrt_ex_osc/Signal Generator', 'Amplitude')

ans =

     2
```

Apply ConvertToTarget and ConvertToComponent Properties

The `ConvertToComponent` transforms the real-time application value into something that can be displayed by the knob. In this example, the strings `"High"`, `"Medium"`, and `"Low"`. The `ConvertToTarget` transforms the knob values into something that can be written to the real-time application. In this example, the values `10`, `5`, and `1`.

This was generated by App Generator.

```
function out_val = convertLabelToValue(in_val, values, labels)
    out_val = values(strcmp(in_val, labels));
end

function out_val = convertValueToLabel(in_val, values, labels)
    out_val = labels(arrayfun(@(x)isequal(in_val, x), values));
end

Signal_Generator_Amplitude_values = [10 5 1];
Signal_Generator_Amplitude_labels = {'High','Medium','Low'};

slrtcomp = slrealtime.ui.tool.ParameterTuner(app.UIFigure, ...
                                             'TargetSource', ...
                                             targetSelector);
slrtcomp.Component = app.Signal_Generator_Amplitude;
slrtcomp.BlockPath = 'slrt_ex_osc/Signal Generator';
slrtcomp.ParameterName = 'Amplitude';
slrtcomp.ConvertToComponent = @(val)app.convertValueToLabel(val, ...
                                    Signal_Generator_Amplitude_values, ...
                                    Signal_Generator_Amplitude_labels);
slrtcomp.ConvertToTarget = @(val)app.convertLabelToValue(val, ...
                                    Signal_Generator_Amplitude_values, ...
                                    Signal_Generator_Amplitude_labels);
```

Input Arguments

`hFigure` — Handle to `uifigure` object

object handle

The `hFigure` argument identifies the `uifigure` to which you are adding the UI component.

Example: `hFig = uifigure()`

Data Types: `function_handle`

Output Arguments

`hPTuner` — Handle to parameter tuner component

component handle

The `hPTuner` argument is the handle to the parameter tuner component that you create for the App Designer component.

Version History

Introduced in R2021b

See Also

`ConnectButton` | `LoadButton` | `InstrumentManager` | `Menu` | `ParameterTable` | `RebootButton` | `RecordButton` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton` | slrealtime.ui Properties

Topics
“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

slrealtime.ui Properties

slrealtime UI component properties for instrument panel UI

Description

Using these properties, you can customize the appearance and operation of `slrealtime.ui.control` controls and `slrealtime.ui.tool` tools.

Properties

Component Operation

`Enable` — Enable component operation

`true` (default) | `false`

This property is available for each of the Simulink Real-Time components. The property selects whether the component is enabled (true) or disabled (false) when the app runs. For graphical (non Simulink Real-Time) components that are configured with a Simulink Real-Time `ParameterTuner`, you can use the `Enable` property on the `ParameterTuner` to enable or disable the graphical component.

- When true, the `ParameterTuner` sets or clears `Enable` property of the graphical component based on state of target computer.
- When false, the `ParameterTuner` sets the `Enable` property of the `ParameterTuner` component to false but continues to update the graphical component. In this way, the `ParameterTuner` `Enable` property acts as an enable property for the graphical component.

Example: `k = uiknob; pt = slrealtime.ui.tool.ParameterTuner; pt.Component = k; pt.Enable = false;`

Data Types: `logical`

Position and Size

`Position` — Location and size of component

`[100 100 200 30]` (default) | [left bottom width height]

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `h.Position = [0 0 200 200]`

Data Types: `integer array`

`IconAlignment` — Location of icon relative to button text

`'left'` (default) | `'right'` | `'center'` | `'top'` | `'bottom'`

This property applies to the `slrealtime.ui.control` button controls. For more information, see the property description in Button Properties.

Example: `'left'`

Data Types: `string`

`HorizontalAlignment` — Horizontal alignment of icon and text

`'center'` (default) | `'left'` | `'right'`

This property applies to the `slrealtime.ui.control` button controls. For more information, see the property description in Button Properties.

Example: `'center'`

Data Types: `string`

`VerticalAlignment` — Vertical alignment of icon and text

`'center'` (default) | `'top'` | `'bottom'`

This property applies to the `slrealtime.ui.control` button controls. For more information, see the property description in Button Properties.

Example: `'center'`

Data Types: `string`

Component Appearance

`FontName` — Font name

`'Helvetica'` (default) | system supported font name

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `'Helvetica'`

Data Types: `system supported font name`

`FontSize` — Font size

12 (default) | positive number

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: 12

Data Types: `positive number`

`FontWeight` — Font weight

`'normal'` (default) | `'bold'`

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `'normal'`

Data Types: `string`

`FontAngle` — Selects font angle for component text

`'normal'` (default) | string

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `'normal'`

Data Types: `string`

`FontColor` — Selects
`0 0 0` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'`

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `0 0 0`

Data Types: `RGB triplet`

`BackgroundColor` — Background color
`[0.96 0.96 0.96]` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'`

This property applies to all `slrealtime.ui.control` controls. For more information, see the property description in UI Figure Properties.

Example: `[0.96 0.96 0.96]`

Data Types: `RGB triplet`

Connect Button Component

`ConnectedIcon` — Icon for button
`'slrtConnectIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.ConnectButton` component. The property selects the icon that is displayed on the button in the connected state.

Example: `'slrtConnectIcon.png'`

Data Types: `string`

`DisconnectedIcon` — Icon for button
`'slrtDisconnectIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.ConnectButton` component. The property selects the icon that is displayed on the button in the disconnected state.

Example: `'slrtDisconnectIcon.png'`

Data Types: `string`

`ConnectedText` — Text for button
`Connected` (default) | string

This property applies to the `slrealtime.ui.control.ConnectButton` component. The property selects the text that is displayed on the button in the connected state.

Example: `'Connected'`

Data Types: `string`

`DisconnectedText` — Text for button
`Disconnected` (default) | string

This property applies to the `slrealtime.ui.control.ConnectButton` component. The property selects the text that is displayed on the button in the disconnected state.

Example: `'Disconnected'`

Data Types: `string`

Load Button Component

`ShowLoadedApplication` — Show loaded application
`true` (default) | `false`

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects whether the button displays the name of the loaded real-time application in the loaded state.

Example: `true`

Data Types: `logical`

`LoadIcon` — Icon for button
`'slrtLoadIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtLoadIcon.png'`

Data Types: `string`

`LoadText` — Text for button
`'Load Application'` (default) | string

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects the text that is displayed on the button.

Example: `'Load Application'`

Data Types: `string`

`SkipInstall` — Skip installation of application
`false` (default) | `true`

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects whether the button action skips installation and just loads the real-time application.

Example: `false`

Data Types: `logical`

`AsyncLoad` — Asynchronous load of application
`false` (default) | `true`

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects whether the button action that loads the real-time application is asynchronous (does not block) MATLAB operation.

Example: `false`

Data Types: `logical`

`Application` — Application to load
empty (default) | application name

This property applies to the `slrealtime.ui.control.LoadButton` component. The property selects the application that loads when you click the button. If set, the button loads the specified application instead of opening a dialog box to select an application.

Example: `'myApplication'`

Data Types: `string`

Instrument Manager Component

`Instruments` — Managed Instrument objects

empty (default) | array of Instrument objects

This property applies to the `slrealtime.ui.tool.InstrumentManager` component. Use this property to add `Instrument` objects to the array of managed instruments. The `Instrument` objects are bound to one or more UI components.

Example: `[hInstAxes hInstGauge]`

Data Types: `array of Instrument objects`

Menu Component

`AsyncLoad` — Asynchronous load of application

`false` (default) | `true`

For property information, see `AsyncLoad` description. For an example applying this property for the `slrealtime.ui.container.Menu` component, see “Create Menu by Using App Generator” on page 1-259

Example: `0`

Data Types: `logical`

`AutoImportFileLog` — Import file log on application stop

`1` (import) (default) | `0` (no import)

For property information, see `AutoImportFileLog` description. For an example applying this property for the `slrealtime.ui.container.Menu` component, see “Create Menu by Using App Generator” on page 1-259

Example: `0`

Data Types: `logical`

`ExportToBaseWorkspace` — Export file log data to model base workspace on application stop

`1` (export) (default) | `0` (no export)

For property information, see `ExportToBaseWorkspace` description. For an example applying this property for the `slrealtime.ui.container.Menu` component, see “Create Menu by Using App Generator” on page 1-259

Example: `0`

Data Types: `logical`

`ReloadOnStop` — Reload application on stop

`1`(reload) (default) | `0` (no reload)

For property information, see `ReloadOnStop` description. For an example applying this property for the `slrealtime.ui.container.Menu` component, see “Create Menu by Using App Generator” on page 1-259

Example: `1`

Data Types: `logical`

`SkipInstall` — Skip installation of application

`false` (default) | `true`

For property information, see `SkipInstall` description. For an example applying this property for the `slrealtime.ui.container.Menu` component, see “Create Menu by Using App Generator” on page 1-259

Example: `false`

Data Types: `logical`

Parameter Table and Signal Table Component

`TableForegroundColor` — Table foreground color

`[0 0 0]` (default) | RGB triplet

This property applies to the `slrealtime.ui.control.ParameterTable` and `slrealtime.ui.control.SignalTable` controls. For more information, see the RGB triplet table the property description in UI Figure Properties.

Example: `[0 0 0]`

Data Types: `RGB triplet`

`TableBackgroundColor` — Table background color

`[1 1 1]` (default) | RGB triplet

This property applies to the `slrealtime.ui.control.ParameterTable` and `slrealtime.ui.control.SignalTable` controls. For more information, see the RGB triplet table in the property description in UI Figure Properties.

Example: `[1 1 1]`

Data Types: `RGB triplet`

`Signals` — Signals in signal table

struct

This property applies to the `slrealtime.ui.control.SignalTable` control. The `Signals` property value is a struct that contains fields:

- `BlockPath` — provides either the full block path or name for each signal.
- The `PortIndex` — provides either the port index value (for signals identified by block path) or the value `-1` (for signals identified by name).
- `Decimation` — provides a decimation value for each signal. Default values is `1`.
- `ArrayIndex` — provides an array index for each signal. Default values is `[]`.
- `BusElement` — provides bus element selection for each signal. Default values is `[]`, `''`, or `""`.
- `Callback` — provides callback code for each signal. Default values is `[]`.

```
hSTable.Signals = struct( ...
  'BlockPath', {'testmodel/Constant1', 'testmodel/Constant2', ...
                'Sine', 'String', 'testmodel/Switch1'}, ...
  'PortIndex', {1, 1, -1, -1, 1}, . . .
  'Decimation', {1, 2, 5, 1, 1}, . . .
  'ArrayIndex', {[], [], 2, [2 2], 3}, . . .
  'BusElement', {'', 'b', '', 'z', 'a'}, . . .
  'Callback', {[], cb, [], [], []});
```

Example: `struct`

Data Types: `struct`

`Parameters` — Parameters in parameter table

struct

This property applies to the `slrealtime.ui.control.ParameterTable` control. The `Parameters` property value is a struct that contains fields `BlockPath` and `ParameterName`. The `BlockPath` field provides the full block path for each parameter in the table. The `ParameterName` field provides the parameter name for each parameter in the table. When using workspace variables in the `Parameters` property, the `BlockPath` is empty for the workspace variable, and the `ParameterName` value is the name of the workspace variable.

```
hPTable.Parameters = struct( ...
  'BlockPath', {'testmodel/Constant1', '', ...
         'testmodel/Constant5', '', ...
         'testmodel/str1', 'testmodel/str2', ...
         'testmodel/multi-line block name', ...
         'testmodel/Constant6'}, ...
  'ParameterName', {'Value', 'model_wksp_var', ...
           'Value', 'base_wksp_var', 'String', ...
           'String', 'Value', 'Value'});
```

Example: `struct`

Data Types: `struct`

Parameter Tuner Component

`Component` — App Designer component

empty (default) | graphic object

This property applies to the `slrealtime.ui.tool.ParameterTuner` component. The property identifies the underlying App Designer component connected to the parameter tuner.

Example: `hKnob`

Data Types: `graphic object`

`BlockPath` — Parameter block path

empty (default) | string

This property applies to the `slrealtime.ui.tool.ParameterTuner` component. The property identifies a parameter that is specified by block path and parameter name.

Example: `'testmodel/Constant6'`

Data Types: `string`

`ParameterName` — Parameter name

empty (default) | string

This property applies to the `slrealtime.ui.tool.ParameterTuner` component. The property identifies a parameter that is specified by block path and parameter name.

Example: `'Value'`

Data Types: `string`

`ConvertToComponent` — Convert parameter value to component value

`@app.convToDouble` (default) | handle

This property applies to the `slrealtime.ui.tool.ParameterTuner` component. Use this property to convert a parameter value to a value used by the component `Value` property. For example, you can convert a fixed-point value to a double. For a code example, see “Apply ConvertToTarget and ConvertToComponent Properties” on page 1-293.

Example: `@app.convToDouble`

Data Types: `object handle`

`ConvertToTarget` — Convert component value to target parameter

empty (default) | handle

This property applies to the `slrealtime.ui.tool.ParameterTuner` component. Use this property to convert a component value to a value used by the real-time application on the target computer. For example, you can convert discrete knob states to integer values. For a code example, see “Apply ConvertToTarget and ConvertToComponent Properties” on page 1-293.

Example: `@app.convertedInt`

Data Types: `object handle`

Reboot Button Component

`WaitForReboot` — Pause instrument panel until target computer completes reboot

`1` (pause) (default) | `0` (no pause)

This property applies to the `slrealtime.ui.control.RebootButton` component. The property selects whether to pause operation of the instrument panel while the target computer completes the reboot process.

Example: `0`

Data Types: `logical`

Record Button Component

`StartRecordingIcon` — Icon for button

`'slrtStartRecordIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.RecordButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtStartRecordIcon.png'`

Data Types: `string`

StartRecordingText — Text for button
`'Start Recording'` (default) | string

This property applies to the `slrealtime.ui.control.RecordButton` component. The property selects the text that is displayed on the button.

Example: `'Start Recording'`

Data Types: `string`

StopRecordingIcon — Icon for button
`'slrtStopRecordIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.RecordButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtStopRecordIcon.png'`

Data Types: `string`

StopRecordingText — Text for button
`'Stop Recording'` (default) | string

This property applies to the `slrealtime.ui.control.RecordButton` component. The property selects the text that is displayed on the button.

Example: `'Stop Recording'`

Data Types: `string`

Start-Stop Button Component

AutoImportFileLog — Import file log on application stop
`1` (import) (default) | `0` (no import)

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects whether the button stop action imports the file log from the real-time application.

Example: `1`

Data Types: `logical`

ExportToBaseWorkspace — Export file log data to model base workspace on application stop
`1` (export) (default) | `0` (no export)

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects whether the button stop action exports the file log data to the model base workspace from the real-time application.

Example: `1`

Data Types: `logical`

ReloadOnStop — Reload application on stop
`1`(reload) (default) | `0` (no reload)

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects whether the button stop action reloads the real-time application.

Example: `1`

Data Types: `logical`

`StartIcon` — Icon for button

`'slrtRunIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtRunIcon.png'`

Data Types: `string`

`StartText` — Text for button

`'Start'` (default) | string

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects the text that is displayed on the button.

Example: `'Start'`

Data Types: `string`

`StopIcon` — Icon for button

`'slrtStopIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtStopIcon.png'`

Data Types: `string`

`StopText` — Text for button

`'Stop'` (default) | string

This property applies to the `slrealtime.ui.control.StartStopButton` component. The property selects the text that is displayed on the button.

Example: `'Stop'`

Data Types: `string`

System Log Component

`IncludeTimeStamps` — Include time stamps for log entries

`false` (default) | `true`

This property applies to the `slrealtime.ui.control.SystemLog` component. The property selects whether the log entries include timestamps.

Example: `false`

Data Types: `logical`

Target Computer Selection

`TargetName` — Target computer selected by TargetSelector component (read-only)

default target computer (default) | string

When you select a target computer from the target computer selector component, the component updates its `TargetName` property and synchronizes the `TargetSource` property of all instrument panel controls to the current state of the selection. See the `TargetSource` property.

Example: `myTarget = h.TargetName`

Data Types: `string`

`TargetSource` — Target computer used by this component (write-only)
empty (default) | string | `slrealtime.ui.control.TargetSelector`

When you select a target computer from the target computer selector component, all instrument panel controls synchronize and update to the current state of the selection. All Simulink Real-Time components (other than the `TargetSelector` component) have a `TargetSource` property that has one of these values:

- empty (default)

  Empty is the default and tells the component to use the default SLRT target computer.
- string

  String is the name of an SLRT target computer.
- `slrealtime.ui.control.TargetSelector`

  `slrealtime.ui.control.TargetSelector` enables a component to query the currently selected target and to be notified when the selection changes.

Example: `h.TargetName = 'TargetPC1'`

Data Types: `string` | `slrealtime.ui.control.TargetSelector`

Update Button Component

`UpdateIcon` — Icon for button
`'slrtUpdateIcon.png'` (default) | string

This property applies to the `slrealtime.ui.control.UpdateButton` component. The property selects the icon that is displayed on the button.

Example: `'slrtUpdateIcon.png'`

Data Types: `string`

`UpdateText` — Text for button
`'Update Software'` (default) | string

This property applies to the `slrealtime.ui.control.UpdateButton` component. The property selects the text that is displayed on the button.

Example: `'Update Software'`

Data Types: `string`

Version History

Introduced in R2021b

R2023a: Added properties for components

The App Designer Simulink Real-Time components have an `Enable` property. You can use this property to disable selected components when the app runs. For more information, see the Enable property. For graphical (non Simulink Real-Time) components that are configured with a Simulink Real-Time `ParameterTuner`, you can use the `Enable` property on the `ParameterTuner` to enable or disable the graphical component.

R2022b: Added properties for components

Added properties for these components:

- `Menu` component — added `AsyncLoad` property, `AutoImportFileLog` property, `ExportToBaseWorkspace` property, `ReloadOnStop` property, and `SkipInstall` property.
- `StartStopButton` component — added `AutoImportFileLog` property, `ExportToBaseWorkspace` property, and `ReloadOnStop` property.
- `RebootButton` component — added `WaitForReboot` property.
- `RecordButton` component — added `StartRecordingIcon` property, `StopRecordingIcon` property, `StartRecordingText` property, and `StopRecordingText` property.

See Also

`ConnectButton` | `InstrumentManager` | `LoadButton` | `Menu` | `ParameterTable` | `ParameterTuner` | `SignalTable` | `SimulationTimeEditField` | `StartStopButton` | `StatusBar` | `StopTimeEditField` | `SystemLog` | `TETMonitor` | `TargetSelector` | `UpdateButton`

Topics

“Create App Designer Instrument Panels by Using Simulink Real-Time Components”

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 subordinate device synchronization deviation received
(     8): DCL initialized
(     9): DCM inSync
(    21): Successful subordinate device state transition.
(   100): Queue raw command response notification
( 65537): Cyclic command: Working count error
( 65538): Main device init command: Working count error
( 65539): Subordinate device 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): Subordinate device init command: No response
( 65548): Main device init command: No response
( 65550): Timeout when waiting for mailbox init command response
( 65551): Cyclic command: Not all subordinate devices in op state
( 65552): Ethernet link (cable) not connected
( 65554): Redundancy: Line break detected
( 65555): Cyclic command: A subordinate device is in error state
( 65556): Subordinate device error status change
( 65557): Station address lost (or subordinate device 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 ...
          ecatConfigureMdevice by other thread (RAS)
( 65563): Redundancy: Line is repaired
( 65564): FOE mbox abort
( 65565): Invalid mail box data received
( 65566): PDI watchdog expired on subordinate device, thrown by IST
( 65567): Subordinate device not supported (if redundancy is activated and ...
          subordinate device doesn't fully support autoclose
( 65568): Subordinate device in unexpected state
( 65569): All subordinate 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): Subordinate device disappears
(262150): Subordinate device 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.

1. In your model, connect the output of the EtherCAT Get Notifications block connects to a File Log block.
2. Build the model, and then download and run the real-time application.
3. 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]`.
4. 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`.
5. To get the `timelog` and the `datalog` use:

   ```
   timelog = notifications{1}.Values.Time;
   datalog = notifications{1}.Values.Data;
   ```
6. 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 subordinate device 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 1-309.

```
[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 subordinate device 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 1-309 .

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 1-309 .

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.

Version History

Introduced in R2020b

See Also

EtherCAT Get Notifications

slrealtime.EtherCAT.getSignalNames

Package: `slrealtime`

Display EtherCAT notifications in human-readable format

Syntax

```
[input,output,SubDevices] = slrealtime.EtherCAT.getSignalNames(devID,
modelName)
```

Description

`[input,output,SubDevices] = slrealtime.EtherCAT.getSignalNames(devID, modelName)` gets the PDO input variable names, PDO output variable names, and subordinate device names for a specified device ID in the model. You can use this information to configure the EtherCAT blocks in the model by using `setparam` commands.

Examples

Get EtherCAT Signal Names

Get the PDO input variable names, PDO output variable names, and subordinate device names for a specified device ID in the model `slrt_ex_ethercat_beckhoff_aio`. This example sets the path to the ENI file for the EtherCAT Init block. This approach lets you refer to ENI files that are not available on the MATLAB path.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime',...
    'examples','slrt_ex_ethercat_beckhoff_aio'));
eniPath = fullfile(matlabroot,'toolbox','slrealtime',...
    'examples','BeckhoffAIOconfig.xml')
set_param('slrt_ex_ethercat_beckhoff_aio/EtherCAT Init',...
    'config_file',eniPath)
slbuild('slrt_ex_ethercat_beckhoff_aio');
[myInput,myOutput,mySubDevices] = ...
    slrealtime.EtherCAT.getSignalNames(0,...
    'slrt_ex_ethercat_beckhoff_aio')
```

```
myInput =

  1×4 string array

  Columns 1 through 2

    "Term 2 (EL3062).A…"    "Term 2 (EL3062).A…"

  Columns 3 through 4

    "Term 2 (EL3062).A…"    "Term 2 (EL3062).A…"
```

```
myOutput =

  1×2 string array

    "Term 3 (EL4002).AO Ou…"    "Term 3 (EL4002).AO Ou…"


mySubDevices =

  1×3 string array

  Columns 1 through 2

    "Term 1 (EK1100)"    "Term 2 (EL3062)"

  Column 3

    "Term 3 (EL4002)"
```

Input Arguments

`devID` — Device ID

integer

The `devID` is the EtherCAT device ID of the device in the model for which signals are found. The device ID is typically 0 when a single EtherCAT network is in use in a model.

Example: `0`

`modelName` — Model name

character vector

The `modelName` is the model from which EtherCAT signals are found. If model argument is omitted, the function uses the current model.

Example: `slrt_ex_ethercat_beckhoff_aio`

Output Arguments

`input` — variables in a PDO read block

array of strings

The `input` is an array of strings with the variables usable in a PDO read block.

`output` — variables in a PDO write block

array of strings

The `output` is an array of strings with the variables usable in a PDO write block.

`SubDevices` — Names of the EtherCAT subordinate devices

array of strings

The `SubDevices` is an array of strings with the names of the EtherCAT subordinate devices in the model for use in the CoE and SoE blocks.

Version History

Introduced in R2020b

See Also

EtherCAT Get Notifications

slrealtime.createEthernetPacketBusObj

Package: `slrealtime`

Created Ethernet packet bus object

Syntax

```
slrealtime.createEthernetPacketBusObj(dataLength)
```

Description

`slrealtime.createEthernetPacketBusObj(dataLength)` creates a bus object to use with the Ethernet Receive block and the Ethernet Send block.

If a bus object with the name already exists, it is assumed to contain the `Length` and `Data` elements and the object is updated such that the size of the `Data` element is set to the maximum of the existing size and the `dataLength` argument

Examples

Create Ethernet Packet Object

To create a `Simulink.Bus` object named `Ethernet_Packet` that has bus elements:

- `Data`: DataType: 'uint8' Size: dataLength x 1
- `Length`: DataType: 'uint16' Size: 1

```
slrealtime.createEthernetPacketBusObj(16);
```

Input Arguments

dataLength — Data length in Ethernet packet

uint16 (in the range 14 - 1514)

The **dataLength** selects the number of bytes in the Ethernet packet.

Example: `16`

Data Types: `uint16`

Version History

Introduced in R2022a

See Also

Objects

`Simulink.Bus`

Tools
Type Editor

Blocks
Ethernet Receive | Ethernet Send

Topics
“Apply 802.1Q VLAN Tag by Using Ethernet Send and Receive Blocks”

createPortConfigureFile

Package: `slrealtime`

Generate configuration file for XIL ports object

Syntax

```
slrealtime.createPortConfigureFile(xmlFilename,ipAddress,appFilepath)
```

Description

`slrealtime.createPortConfigureFile(xmlFilename,ipAddress,appFilepath)` generates an XML file that configures a XIL `MAPort`, `ECUMPort`, and `ECUCPort` object for third-party software, such as ECU-TEST.

Examples

Create Port Configuration

To generate a ports object configuration for for third-party software, such as ECU-TEST, use the `createPortConfigureFile` function.

This function generates an XML file `myConfigureFile.xml` that configures a ports object (for third-party software, such as ECU-TEST) for a target computer at IP address `10.10.10.15` and a real-time application `myModel`.

```
slrealtime.createPortConfigureFile('myConfigureFile.xml','10.10.10.15','myModel')
```

Input Arguments

`xmlFilename` — Configuration file name
character vector

Provides the XML file name for the configuration file.

Example: `'myConfigureFile.xml'`

`ipAddress` — Target computer IP address
character vector of the form `xx.xx.xx.xx`

Provides the IP address of the target computer.

Example: `'10.10.10.15'`

`appFilepath` — Real-time application path
character vector

Provides the path to the real-time application MLDATX file.

Example: `'myModel'`

Version History

Introduced in R2021b

See Also

Topics

“Install the Simulink Real-Time Support Package for ASAM XIL Standard”
“Classes and Methods of ASAM XIL API”

External Websites

ASAM XIL

slrealtime.createUDPPacketBusObj

Package: `slrealtime`

Created UDP packet bus object

Syntax

```
slrealtime.createUDPPacketBusObj(dataLength)
```

Description

`slrealtime.createUDPPacketBusObj(dataLength)` creates a bus object to use with the UDP Receive block and UDP Send block.

If a bus object with the name already exists, it is assumed to contain the `IP_Address`, `IP_Port`, `Length`, and `Data` elements and the object is updated such that the size of the `Data` element is set to the maximum of the existing size and the `dataLength` argument

Examples

Create UDP Packet Object

To create a `Simulink.Bus` object named `UDP_Packet` that has bus elements:

- `IP_Address`: DataType: 'uint8' Size: 4 x 1)
- `IP_Port`: DataType: 'uint16' Size: 1
- `Length`: DataType: 'uint16' Size: 1
- `Data`: DataType: 'uint8' Size: dataLength x 1

```
slrealtime.createUDPPacketBusObj(16);
```

Input Arguments

dataLength — Data length in UDP packet

uint16 (in the range 1 - 65507)

The **dataLength** selects the number of bytes in the UDP packet.

Example: 16

Data Types: `uint16`

Version History

Introduced in R2022a

See Also

Objects
`Simulink.Bus`

Tools
Type Editor

Blocks
UDP Receive | UDP Send

slrealtime.getSupportInfo

Creates `slrealtimeinfo.txt` file that provides information about Simulink Real-Time installation

Syntax

```
slrealtime.getSupportInfo()
slrealtime.getSupportInfo(target_name)
slrealtime.getSupportInfo(model_name)
```

Description

`slrealtime.getSupportInfo()` creates an `slrealtimeinfo.txt` file that provides information about the Simulink Real-Time installation and all connected target computers for MathWorks® support.

`slrealtime.getSupportInfo(target_name)` creates an `slrealtimeinfo-TargetName.txt` file that provides information about the Simulink Real-Time installation and the selected target computer for MathWorks support.

`slrealtime.getSupportInfo(model_name)` creates an `slrealtimeinfo.txt` file that provides information about the Simulink Real-Time installation and all connected target computers for MathWorks support. This option also creates a `model_name_configset.m` file that provides information about the selected model.

Examples

Get Support Information for MathWorks Support

To get support information about the Simulink Real-Time installation, connected target computer `TargetPC1`, and a Simulink Real-Time model `slrt_ex_osc`, open the model and run the `slrealtime.getSupportInfo` command.

1. `slrealtime.getSupportInfo(Target='TargetPC1',Model='slrt_ex_osc');`
2. You also can use this syntax for the command:

   ```
   slrealtime.getSupportInfo('Target','TargetPC1','Model','slrt_ex_osc');
   ```

Input Arguments

`target_name` — Target name

character vector | string scalar

Provides name of a selected target computer for which you are building a real-time application. If using a target object name instead of a target computer name character vector, omit quotes. For example, `Target=tg`.

Example: `Target='TargetPC1'`

Example: `Target=tg`

Example: `'Target','TargetPC1'`

`model_name` — Model name
character vector | string scalar

Provides name of Simulink Real-Time model from which you are building a real-time application. You can add the SLX extension on the model name.

Example: `Model='slrt_ex_osc'`

Example: `'Model','slrt_ex_osc'`

Version History

Introduced in R2020b

R2023a: Added target and model selection arguments

The `slrealtime.getSupportInfo` function `Target` option lets you select a target computer for which the function gathers support information. The `slrealtime.getSupportInfo` function `Model` option lets you select a Simulink model for which the function gathers support information. For the operation from previous release that gathers support information for all connected target computers, use the syntax `slrealtime.getSupportInfo()` with no options.

See Also

`slrealtime.getCrashStack`

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 represents 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.

Version History

Introduced in R2020b

See Also

`slrealtime.getSupportInfo`

Simulink.sdi.compareRuns

Package: `Simulink.sdi`

Compare data in two simulation runs

Syntax

```
diffResult = Simulink.sdi.compareRuns(runID1,runID2)
diffResult = Simulink.sdi.compareRuns(runID1,runID2,Name=Value)
```

Description

`diffResult = Simulink.sdi.compareRuns(runID1,runID2)` compares the data in the runs that correspond to `runID1` and `runID2` and returns the result in the `Simulink.sdi.DiffRunResult` object `diffResult`. For more information about the comparison algorithm, see “How the Simulation Data Inspector Compares Data”.

`diffResult = Simulink.sdi.compareRuns(runID1,runID2,Name=Value)` compares the simulation runs that correspond to `runID1` and `runID2` using the options specified by one or more name-value arguments. For more information about comparison options, see “How the Simulation Data Inspector Compares Data”.

Examples

Compare Runs with Global Tolerance

You can specify global tolerance values to use when comparing two simulation runs. Global tolerance values are applied to all signals within the run. This example shows how to specify global tolerance values for a run comparison and how to analyze and save the comparison results.

First, load the session file that contains the data to compare. The session file contains data for four simulations of an aircraft longitudinal controller. This example compares data from two runs that use different input filter time constants.

```
Simulink.sdi.load('AircraftExample.mldatx');
```

To access the run data to compare, use the `Simulink.sdi.getAllRunIDs` function to get the run IDs that correspond to the last two simulation runs.

```
runIDs = Simulink.sdi.getAllRunIDs;
runID1 = runIDs(end - 1);
runID2 = runIDs(end);
```

Use the `Simulink.sdi.compareRuns` function to compare the runs. Specify a global relative tolerance value of `0.2` and a global time tolerance value of `0.5`.

```
runResult = Simulink.sdi.compareRuns(runID1,runID2,'reltol',0.2,'timetol',0.5);
```

Check the `Summary` property of the returned `Simulink.sdi.DiffRunResult` object to see whether signals were within the tolerance values or out of tolerance.

```
runResult.Summary
```

```
ans = struct with fields:
       OutOfTolerance: 0
      WithinTolerance: 3
            Unaligned: 0
        UnitsMismatch: 0
                Empty: 0
             Canceled: 0
          EmptySynced: 0
     DataTypeMismatch: 0
         TimeMismatch: 0
    StartStopMismatch: 0
          Unsupported: 0
```

All three signal comparison results fell within the specified global tolerance.

You can save the comparison results to an MLDATX file using the `saveResult` function.

```
saveResult(runResult,'InputFilterComparison');
```

Analyze Simulation Data Using Signal Tolerances

You can programmatically specify signal tolerance values to use in comparisons performed using the Simulation Data Inspector. In this example, you compare data collected by simulating a model of an aircraft longitudinal flight control system. Each simulation uses a different value for the input filter time constant and logs the input and output signals. You analyze the effect of the time constant change by comparing results using the Simulation Data Inspector and signal tolerances.

First, load the session file that contains the simulation data.

```
Simulink.sdi.load('AircraftExample.mldatx');
```

The session file contains four runs. In this example, you compare data from the first two runs in the file. Access the `Simulink.sdi.Run` objects for the first two runs loaded from the file.

```
runIDs = Simulink.sdi.getAllRunIDs;
runIDTs1 = runIDs(end-3);
runIDTs2 = runIDs(end-2);
```

Now, compare the two runs without specifying any tolerances.

```
noTolDiffResult = Simulink.sdi.compareRuns(runIDTs1,runIDTs2);
```

Use the `getResultByIndex` function to access the comparison results for the `q` and `alpha` signals.

```
qResult = getResultByIndex(noTolDiffResult,1);
alphaResult = getResultByIndex(noTolDiffResult,2);
```

Check the `Status` of each signal result to see whether the comparison result fell within our out of tolerance.

```
qResult.Status
```

```
ans =
  ComparisonSignalStatus enumeration
```

```
    OutOfTolerance
```

```
alphaResult.Status
```

```
ans =
  ComparisonSignalStatus enumeration

    OutOfTolerance
```

The comparison used a value of `0` for all tolerances, so the `OutOfTolerance` result means the signals are not identical.

You can further analyze the effect of the time constant by specifying tolerance values for the signals. Specify the tolerances by setting the properties for the `Simulink.sdi.Signal` objects that correspond to the signals being compared. Comparisons use tolerances specified for the baseline signals. This example specifies a time tolerance and an absolute tolerance.

To specify a tolerance, first access the `Signal` objects from the baseline run.

```
runTs1 = Simulink.sdi.getRun(runIDTs1);
qSig = getSignalsByName(runTs1,'q, rad/sec');
alphaSig = getSignalsByName(runTs1,'alpha, rad');
```

Specify an absolute tolerance of `0.1` and a time tolerance of `0.6` for the `q` signal using the `AbsTol` and `TimeTol` properties.

```
qSig.AbsTol = 0.1;
qSig.TimeTol = 0.6;
```

Specify an absolute tolerance of `0.2` and a time tolerance of `0.8` for the `alpha` signal.

```
alphaSig.AbsTol = 0.2;
alphaSig.TimeTol = 0.8;
```

Compare the results again. Access the results from the comparison and check the `Status` property for each signal.

```
tolDiffResult = Simulink.sdi.compareRuns(runIDTs1,runIDTs2);
qResult2 = getResultByIndex(tolDiffResult,1);
alphaResult2 = getResultByIndex(tolDiffResult,2);
```

```
qResult2.Status
```

```
ans =
  ComparisonSignalStatus enumeration

    WithinTolerance
```

```
alphaResult2.Status
```

```
ans =
  ComparisonSignalStatus enumeration

    WithinTolerance
```

Configure Comparisons to Check Metadata

You can use the `Simulink.sdi.compareRuns` function to compare signal data and metadata, including data type and start and stop times. A single comparison may check for mismatches in one or more pieces of metadata. When you check for mismatches in signal metadata, the `Summary` property of the `Simulink.sdi.DiffRunResult` object may differ from a basic comparison because the `Status` property for a `Simulink.sdi.DiffSignalResult` object can indicate the metadata mismatch. You can configure comparisons using the `Simulink.sdi.compareRuns` function for imported data and for data logged from a simulation.

This example configures a comparison of runs created from workspace data three ways to show how the `Summary` of the `DiffSignalResult` object can provide specific information about signal mismatches.

Create Workspace Data

The `Simulink.sdi.compareRuns` function compares time series data. Create data for a sine wave to use as the baseline signal, using the `timeseries` format. Give the `timeseries` the name `Wave Data`.

```
time = 0:0.1:20;
sig1vals = sin(2*pi/5*time);
sig1_ts = timeseries(sig1vals,time);
sig1_ts.Name = 'Wave Data';
```

Create a second sine wave to compare against the baseline signal. Use a slightly different time vector and attenuate the signal so the two signals are not identical. Cast the signal data to the `single` data type. Also name this `timeseries` object `Wave Data`. The Simulation Data Inspector comparison algorithm will align these signals for comparison using the name.

```
time2 = 0:0.1:22;
sig2vals = single(0.98*sin(2*pi/5*time2));
sig2_ts = timeseries(sig2vals,time2);
sig2_ts.Name = 'Wave Data';
```

Create and Compare Runs in the Simulation Data Inspector

The `Simulink.sdi.compareRuns` function compares data contained in `Simulink.sdi.Run` objects. Use the `Simulink.sdi.createRun` function to create runs in the Simulation Data Inspector for the data. The `Simulink.sdi.createRun` function returns the run ID for each created run.

```
runID1 = Simulink.sdi.createRun('Baseline Run','vars',sig1_ts);
runID2 = Simulink.sdi.createRun('Compare to Run','vars',sig2_ts);
```

You can use the `Simulink.sdi.compareRuns` function to compare the runs. The comparison algorithm converts the signal data to the `double` data type and synchronizes the signal data before computing the difference signal.

```
basic_DRR = Simulink.sdi.compareRuns(runID1,runID2);
```

Check the `Summary` property of the returned `Simulink.sdi.DiffRunResult` object to see the result of the comparison.

```
basic_DRR.Summary
```

```
ans = struct with fields:
       OutOfTolerance: 1
      WithinTolerance: 0
            Unaligned: 0
        UnitsMismatch: 0
                Empty: 0
             Canceled: 0
          EmptySynced: 0
     DataTypeMismatch: 0
         TimeMismatch: 0
    StartStopMismatch: 0
          Unsupported: 0
```

The difference between the signals is out of tolerance.

Compare Runs and Check for Data Type Match

Depending on your system requirements, you may want the data types for signals you compare to match. You can use the `Simulink.sdi.compareRuns` function to configure the comparison algorithm to check for and report data type mismatches.

```
dataType_DRR = Simulink.sdi.compareRuns(runID1,runID2,'DataType','MustMatch');
dataType_DRR.Summary
```

```
ans = struct with fields:
       OutOfTolerance: 0
      WithinTolerance: 0
            Unaligned: 0
        UnitsMismatch: 0
                Empty: 0
             Canceled: 0
          EmptySynced: 0
     DataTypeMismatch: 1
         TimeMismatch: 0
    StartStopMismatch: 0
          Unsupported: 0
```

The result of the signal comparison is now `DataTypeMismatch` because the data for the baseline signal is `double` data type, while the data for the signal compared to the baseline is `single` data type.

Compare Runs and Check for Start and Stop Time Match

You can use the `Simulink.sdi.compareRuns` function to configure the comparison algorithm to check whether the aligned signals have the same start and stop times.

```
startStop_DRR = Simulink.sdi.compareRuns(runID1,runID2,'StartStop','MustMatch');
startStop_DRR.Summary
```

```
ans = struct with fields:
       OutOfTolerance: 0
      WithinTolerance: 0
            Unaligned: 0
        UnitsMismatch: 0
```

```
                Empty: 0
             Canceled: 0
          EmptySynced: 0
       DataTypeMismatch: 0
           TimeMismatch: 0
      StartStopMismatch: 1
            Unsupported: 0
```

The signal comparison result is now `StartStopMismatch` because the signals created in the workspace have different stop times.

Compare Runs with Alignment Criteria

When you compare runs using the Simulation Data Inspector, you can specify alignment criteria that determine how signals are paired with each other for comparison. This example compares data from simulations of a model of an aircraft longitudinal control system. The simulations used a square wave input. The first simulation used an input filter time constant of `0.1s` and the second simulation used an input filter time constant of `0.5s`.

First, load the simulation data from the session file that contains the data for this example.

```
Simulink.sdi.load('AircraftExample.mldatx');
```

The session file contains data for four simulations. This example compares data from the first two runs. Access the run IDs for the first two runs loaded from the session file.

```
runIDs = Simulink.sdi.getAllRunIDs;
runIDTs1 = runIDs(end-3);
runIDTs2 = runIDs(end-2);
```

Before running the comparison, define how you want the Simulation Data Inspector to align the signals between the runs. This example aligns signals by their name, then by their block path, and then by their Simulink identifier.

```
alignMethods = [Simulink.sdi.AlignType.SignalName
                Simulink.sdi.AlignType.BlockPath
                Simulink.sdi.AlignType.SID];
```

Compare the simulation data in your two runs, using the alignment criteria you specified. The comparison uses a small time tolerance to account for the effect of differences in the step size used by the solver on the transition of the square wave input.

```
diffResults = Simulink.sdi.compareRuns(runIDTs1,runIDTs2,'align',alignMethods,...
    'timetol',0.005);
```

You can use the `getResultByIndex` function to access the comparison results for the aligned signals in the runs you compared. You can use the `Count` property of the `Simulink.sdi.DiffRunResult` object to set up a `for` loop to check the `Status` property for each `Simulink.sdi.DiffSignalResult` object.

```
numComparisons = diffResults.count;

for k = 1:numComparisons
    resultAtIdx = getResultByIndex(diffResults,k);
```

```
    sigID1 = resultAtIdx.signalID1;
    sigID2 = resultAtIdx.signalID2;

    sig1 = Simulink.sdi.getSignal(sigID1);
    sig2 = Simulink.sdi.getSignal(sigID2);

    displayStr = 'Signals %s and %s: %s \n';
    fprintf(displayStr,sig1.Name,sig2.Name,resultAtIdx.Status);
end
```

```
Signals q, rad/sec and q, rad/sec: OutOfTolerance
Signals alpha, rad and alpha, rad: OutOfTolerance
Signals Stick and Stick: WithinTolerance
```

Input Arguments

`runID1` — Baseline run identifier

integer

Numeric identifier for the baseline run in the comparison, specified as a run ID that corresponds to a run in the Simulation Data Inspector. The Simulation Data Inspector assigns run IDs when runs are created. You can get the run ID for a run by using the `ID` property of the `Simulink.sdi.Run` object, the `Simulink.sdi.getAllRunIDs` function, or the `Simulink.sdi.getRunIDByIndex` function.

`runID2` — Identifier for run to compare

integer

Numeric identifier for the run to compare, specified as a run ID that corresponds to a run in the Simulation Data Inspector. The Simulation Data Inspector assigns run IDs when runs are created. You can get the run ID for a run by using the `ID` property of the `Simulink.sdi.Run` object, the `Simulink.sdi.getAllRunIDs` function, or the `Simulink.sdi.getRunIDByIndex` function.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `AbsTol=x,Align=alignOpts`

`Align` — Signal alignment options

`Simulink.sdi.AlignType` scalar | `Simulink.sdi.AlignType` vector

Signal alignment options, specified as a `Simulink.sdi.AlignType` scalar or vector. The `Simulink.sdi.AlignType` enumeration includes a value for each option available for pairing each signal in the baseline run with a signal in the comparison run. You can specify one or more alignment options for the comparison. To use more than one alignment option, specify an array. When you specify multiple alignment options, the Simulation Data Inspector aligns signals first by the option in the first element of the array, then by the option in the second element array, and so on. For more information, see “Signal Alignment”.

Value	Aligns By
`Simulink.sdi.AlignType.BlockPath`	Path to the source block for the signal
`Simulink.sdi.AlignType.SID`	Automatically assigned Simulink identifier
`Simulink.sdi.AlignType.SignalName`	Signal name
`Simulink.sdi.AlignType.DataSource`	Path of the variable in the MATLAB workspace

Example: `[Simulink.sdi.AlignType.SignalName,Simulink.sdi.AlignType.BlockPath]` specifies signal alignment by signal name and then by block path.

`AbsTol` — Global absolute tolerance for comparison
`0` (default) | positive-valued scalar

Global absolute tolerance for comparison, specified as a positive-valued scalar.

Global tolerances apply to all signals in the run comparison. To use a different tolerance value for a signal in the comparison, specify the tolerance you want to use on the `Simulink.sdi.Signal` object in the baseline run and set the `OverrideGlobalTol` property for that signal to `true`.

For more information about how tolerances are used in comparisons, see “Tolerance Specification”.

Example: `0.5`

Data Types: `double`

`RelTol` — Global relative tolerance for comparison
`0` (default) | positive-valued scalar

Global relative tolerance for comparison, specified as a positive-valued scalar. The relative tolerance is expressed as a fractional multiplier. For example, `0.1` specifies a 10 percent tolerance.

Global tolerances apply to all signals in the run comparison. To use a different tolerance value for a signal in the comparison, specify the tolerance you want to use on the `Simulink.sdi.Signal` object in the baseline run and set the `OverrideGlobalTol` property for that signal to `true`.

For more information about how tolerances are used in comparisons, see “Tolerance Specification”.

Example: `0.1`

Data Types: `double`

`TimeTol` — Global time tolerance for comparison
`0` (default) | positive-valued scalar

Global time tolerance for comparison, specified as a positive-valued scalar, using units of seconds.

Global tolerances apply to all signals in the run comparison. To use a different tolerance value for a signal in the comparison, specify the tolerance you want to use on the `Simulink.sdi.Signal` object in the baseline run and set the `OverrideGlobalTol` property for that signal to `true`.

For more information about tolerances in the Simulation Data Inspector, see “Tolerance Specification”.

Example: `0.2`

Data Types: `double`

`DataType` — Comparison sensitivity to signal data types
`"MustMatch"`

Comparison sensitivity to signal data types, specified as `"MustMatch"`. Specify `DataType="MustMatch"` when you want the comparison to be sensitive to numeric data type mismatches in compared signals.

When signal data types do not match, the `Status` property of the `Simulink.sdi.DiffSignalResult` object for the result is set to `DataTypeMismatch`.

The `Simulink.sdi.compareRuns` function compares the data types for aligned signals before synchronizing and comparing the signal data. When you do not specify this name-value argument, the comparison checks data types only to detect a comparison between string and numeric data. For a comparison between string and numeric data, results are not computed, and the status for the result is `DataTypeMismatch`. For aligned signals that have different numeric data types, the comparison computes results.

When you configure the comparison to stop on the first mismatch, a data type mismatch stops the comparison. A stopped comparison may not compute results for all signals.

`Time` — Comparison sensitivity to signal time vectors
`"MustMatch"`

Comparison sensitivity to signal time vectors, specified as `"MustMatch"`. Specify `Time="MustMatch"` when you want the comparison to be sensitive to mismatches in the time vectors of compared signals. When you specify this name-value argument, the algorithm compares the time vectors of aligned signals before synchronizing and comparing the signal data.

When the time vectors for signals do not match, the `Status` property of the `Simulink.sdi.DiffSignalResult` object for the result is set to `TimeMismatch`.

Comparisons are not sensitive to differences in signal time vectors unless you specify this name-value argument. For comparisons that are not sensitive to differences in the time vectors, the comparison algorithm synchronizes the signals prior to the comparison. For more information about how synchronization works, see “How the Simulation Data Inspector Compares Data”.

When you specify that time vectors must match and configure the comparison to stop on the first mismatch, a time vector mismatch stops the comparison. A stopped comparison may not compute results for all signals.

`StartStop` — Comparison sensitivity to signal start and stop times
`"MustMatch"`

Comparison sensitivity to signal start and stop times, specified as `"MustMatch"`. Specify `StartStop="MustMatch"` when you want the comparison to be sensitive to mismatches in signal start and stop times. When you specify this name-value argument, the algorithm compares the start and stop times for aligned signals before synchronizing and comparing the signal data.

When the start times and stop times do not match, the `Status` property of the `Simulink.sdi.DiffSignalResult` object for the result is set to `StartStopMismatch`.

When you specify that start and stop times must match and configure the comparison to stop on the first mismatch, a start or stop time mismatch stops the comparison. A stopped comparison may not compute results for all signals.

`StopOnFirstMismatch` — Whether comparison stops on first detected mismatch
`"Metadata"` | `"Any"`

Whether comparison stops on first detected mismatch without comparing remaining signals, specified as `"Metadata"` or `"Any"`. A stopped comparison may not compute results for all signals, and can return a mismatched result more quickly.

- `"Metadata"` — A mismatch in metadata for aligned signals stops the comparison. Metadata comparisons happen before comparing signal data.

  The Simulation Data Inspector always aligns signals and compares signal units. When you configure the comparison to stop on the first mismatch, an unaligned signal or mismatched units always stop the comparison. You can specify additional name-value arguments to configure the comparison to check and stop on the first mismatch for additional metadata, such as signal data type, start and stop times, and time vectors.
- `"Any"` — A mismatch in metadata or signal data for aligned signals stops the comparison.

`ExpandChannels` — Whether to compute comparison results for each channel in multidimensional signals
`true` or `1` (default) | `false` or `0`

Whether to compute comparison results for each channel in multidimensional signals, specified as logical `true` (`1`) or `false` (`0`).

- `true` or `1` — Comparison expands multidimensional signals represented as a single signal with nonscalar sample values to a set of signals with scalar sample values and computes a comparison result for each signal.

  The representation of the multidimensional signal in the Simulation Data Inspector as a single signal with nonscalar sample values does not change.
- `false` or `0` — Comparison does not compute results for multidimensional signals represented as a single signal with nonscalar sample values.

Output Arguments

`diffResult` — Comparison results
`Simulink.sdi.DiffRunResult` object

Comparison results, returned as a `Simulink.sdi.DiffRunResult` object.

Limitations

The Simulation Data Inspector does not support comparing:

- Signals of data types `int64` or `uint64`.
- Variable-size signals.

Version History

Introduced in R2011b

See Also

Functions
`Simulink.sdi.compareSignals` | `Simulink.sdi.getRunIDByIndex` | `Simulink.sdi.getRunCount` | `getResultByIndex`

Objects
`Simulink.sdi.DiffRunResult` | `Simulink.sdi.DiffSignalResult`

Topics
“Inspect and Compare Data Programmatically”
“Compare Simulation Data”
“How the Simulation Data Inspector Compares Data”

slrealtime.exportRun

Package: `slrealtime`

Access data for Simulation Data Inspector run

Syntax

```
dataSet = slrealtime.exportRun(runID)
dataSet = slrealtime.exportRun(runID,Name=Value)
```

Description

`dataSet = slrealtime.exportRun(runID)` exports the data from the **Simulation Data Inspector** run corresponding to `runID`.

`dataSet = slrealtime.exportRun(runID,Name=Value)` specifies additional options using one or more `name-value` pair arguments.

Note The `slrealtime.exportRun` function is very similar to the `Simulink.sdi.exportRun` function with few arguments. Use the `slrealtime.exportRun` function for deployed applications.

Examples

Export File Log Data for Selected Run

Open the `slrt_ex_osc` model.

In the Simulink Editor, on the **Real-Time** tab, click **Hardware Settings**.

In the **Simulink Real-Time Options** pane, change **Max file log runs** to 5 and click **OK**.

Click **Run on Target**.

Navigate to the parent of the applications folder and list the file logs that are available to export from the **Simulation Data Inspector**.

```
runs = Simulink.sdi.getAllRunIDs
run=runs(end-1)
```

Export the file log for the penultimate run from the Simulation Data Inspector to a MAT file named `dataPenultimateRun`.

```
runID=runs(end-1)
dataset=slrealtime.exportRun(runID,to="file",fileName="dataPenultimateRun.mat");
```

Input Arguments

`runID` — Run identifier

positive integer | vector of positive integers

Run identifier specified as a positive integer or vector of positive integers. for the run you want a `.MAT` file for. The Simulation Data Inspector assigns run identifiers when you create runs. Get the identifier for a run using `Simulink.sdi.getAllRunIDs` or `Simulink.sdi.getRunIDByIndex`.

To export data for multiple runs to a file, you can specify the `runID` input as a vector of run IDs.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: filename='file_run.MAT'

`to` — Option to export data to workspace variable or a `.MAT` file

`"variable"` (default) | `"file"`

Option to export data to a workspace variable or a MAT file, specified as `"variable"` or `"file"`.

When you export data to a file, you must also specify a file name using the `filename` name-value argument.

Example: `"file"`

`fileName` — Name of MAT file

character vector | string scalar

Name of `.MAT` file, specified as a character vector or string scalar. The `slrealtime.exportRun` function exports only MAT files and does not support others formats.

Example: `"file_run.MAT"`

Dependency

To enable this argument, specify the `to` argument as `file`.

Output Arguments

`dataSet` — Data for run

variable | `.MAT` file

Data for the run corresponding to the run identifier `runID`, returned as a variable or `.MAT` file.

Version History

Introduced in R2023a

See Also

`slrealtime` | `Simulink.sdi.exportRun`

Topics

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

“Save and Reload Parameters by Using the MATLAB Language”

S-Function Status Log API

log_trace

Trace level status message

Syntax

```
void log_trace(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as trace log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b

log_debug

Debug level status message

Syntax

```
void log_debug(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as debug log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b

log_info

Information level status message

Syntax

```
void log_info(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as info log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b

log_warning

Warning level status message

Syntax

```
void log_warning(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as warning log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b

log_error

Error level status message

Syntax

```
void log_error(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as error log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b

log_fatal

Fatal level status message

Syntax

```
void log_fatal(const std::string & msg)
```

Arguments

`msg`

The message to display in the system log.

Description

Use this function to log the message msg in the system log as fatal log level. For an example, see “Add Custom Messages to System Log” on page 1-216.

Languages

C++

Include

```
#include "slrt_log.hpp"
```

See Also

`ssSetErrorStatus`

Version History

Introduced in R2021b