Real-Time Testing

Library Reference

For Real-Time Testing 1.9 ... 5.0

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication and the contents hereof are subject to change without notice.

Contents

About This Reference 11

Real-Time Test Manager Commands 13

Basic Interface 14
About Real-Time Testing 14
dSPACE Help 15
Controlbars 15
Exit 16
Log Viewer 16
New Features and Migration 18
Reset to Default 19
Status Bar 19
Using dSPACE Help 20

Platform Managing 21
Connect 21
Create Sequence 22
Manage Recent Platform Configuration 23
Refresh Platform Configuration 25
Register Platforms 26

Sequence Managing 31
Continue 31
Copy Error Text 32
Delete 32
Open 33
Pause 34
Reload 34
Run 35
Sequence List 36
Stop 37

Tools 39
Explore Demos Folder 39
General Properties 40
Unzip Demos 41

dSPACE Python Modules for Managing RTT Sequences 43

rttmanagerlib Module.. 44

rttmanagerlib Module Quick Reference... 45

Overview of the rttmanagerlib Object Model... 45

RealTimeTestManagerServer... 47

RealTimeTestManagerServer Class Description... 48

AccessBoard Method... 49

ApplicationRTTVersion Method.. 50

BCGServiceProvider... 52

BCGServiceProvider Class Description.. 52

Generate Method... 53

Board.. 54

Board Class Description... 54

Sequences (Collection).. 55

Sequences Class (Collection) Description... 56

ContinueAll Method... 57

Create Method... 57

Item Method... 59

PauseAll Method.. 60

Remove Method... 60

RunAll Method... 61

StopAll Method.. 61

SequencesEvents.. 62

SequencesEvents Class Description.. 63

OnCreate Method.. 65

OnError Method... 65

OnRemove Method.. 66

OnStateChanged Method.. 67

OnWrite Method.. 67

Sequence.. 68

Sequence Class Description... 69

Continue Method... 70

Pause Method... 71

Remove Method... 71

Run Method... 72

Stop Method.. 73

SequenceEvents.. 74

SequenceEvents Class Description.. 74

OnHostCall Method.. 76

DataStreams.. 77

DataStreams Collection Description... 77

ExecutionError... 78

ExecutionError Class Description... 78

Variables (Collection)... 79

Variables Class (Collection) Description... 79

Item Method.. 80

VariablesEvents... 80

VariablesEvents Class Description... 81

OnAdd Method... 82

Variable... 83

Variable Class Description.. 83

Common Constants... 84

Common Constants of rttmanagerlib.. 84

rttutilities Module.. 86

GetTraceBackString Function... 86

RTTSleep Function Description.. 87

dSPACE Python Modules for Implementing RTT Sequences 89

rttlib.canlib Module... 91

rttlib.canlib Module Quick Reference... 92

Overview of the canlib Object Model.. 92

controllers... 93

controllers Class Description.. 94

canmmbaselib... 94

canmmbaselib Class Description.. 94

canmmlib.. 95

canmmlib Class Description.. 96

GetController Method.. 96

controller... 98

controller Class Description.. 98

GetChannel Method... 99

channel.. 100

channel Class Description.. 100

GetRawMessage Method.. 101

message.. 102

message Class Description.. 102

Receive Method.. 104

ReceiveGen Method.. 105

Transmit Method.. 106

TransmitGen Method.. 106

messagerx.. 107

messagerx Class Description.. 107

messagetx.. 108

messagetx Class Description.. 108

canmmerror.. 109

canmmerror Class Description.. 110

rttlib.dscanapilib Module.. 111

dscanapilib.. 111

Class Description (dscanapilib).. 113

ActivateChannel.. 115

DeactivateChannel.. 116

EnableBusStatistics.. 116

EncodeBusStatistics.. 117

FlushReceiveQueue.. 118

FlushTransmitQueue.. 119

GetAvailableChannels.. 120

GetBaudrate.. 120

GetBusInfo.. 121

GetBusType.. 122

GetErrorText.. 123

GetHardwareTime.. 123

GetHardwareTimeResolution.. 124

InitChannel.. 125

ReadReceiveQueue.. 126

RegisterChannel.. 127

ResetHardwareTime.. 128

SetAcceptance.. 129

SetBaudrate.. 130

SetChannelOutput.. 131

SetTransmitAcknowledge.. 132

TransmitMessages... 132
UnregisterChannel... 133

BitTimingParameters... 134
Class Description (BitTimingParameters)... 134

BusInfo... 135
Class Description (BusInfo)... 135

BusStatistics... 136
Class Description (BusStatistics)... 136

CanMessage... 137
Class Description (CanMessage)... 138

ChannelInfo... 139
Class Description (ChannelInfo)... 139

rttlib.datastream Module... 141

CreateVariableMap... 141
CreateVariableMap Class Description... 142

VariableMap... 143
VariableMap Class Description... 143
AddVariable Method... 144

MatFile... 145
MatFile Class Description... 146
Replay Method... 149

CreateVariableMapMDF... 150
CreateVariableMapMDF Class Description... 150

VariableMapMDF... 152
VariableMapMDF Class Description... 152
AddVariable Method... 153

MDFFile... 154
MDFFile Class Description... 154
Replay Method... 158

rttlib.dynamicvariable Module... 160
DynamicVariable Class... 160
Name Method... 161

rttlib.errors Module... 163

RTTException... 163
RTTException Class Description... 163

rttlib.dsethernetapilib Module........ 164
AccessProviderInfo Class........ 166
InterfaceInfo Class........ 166
EthRawFrameHeader Class........ 167
EthRawFrame Class........ 168
ActivateInterface Method........ 169
CreateBuffer Method........ 170
DeactivateInterface Method........ 170
FlushReceiveQueue Method........ 171
FlushTransmitQueue Method........ 172
GetAccessProviders Method........ 172
GetAccessProvidersCount Method........ 173
GetAvailableInterfaces Method........ 173
GetAvailableInterfacesCount Method........ 174
GetInterfaceCapabilities Method........ 175
GetInterfaceMacAddress Method........ 175
GetSupportedFrameTypes Method........ 176
GetSupportedFrameTypesCount Method........ 177
GetTime Method........ 178
GetTimeResolution Method........ 178
InitInterface Method........ 179
IsInterfaceAccessible Method........ 181
IsInterfaceAvailable Method........ 181
ReadFrames Method........ 182
RegisterInterface Method........ 183
SetFilter Method........ 184
TransmitFrames Method........ 185
UnregisterInterface Method........ 186

rttlib.globalvariables Module........ 187
rttlib.globalvariables Module Description........ 187

rttlib.hostcall Module........ 188
Hostcall Function........ 188

rttlib.rs232lib Module........ 190
OpenEx Function........ 190
SetConfig Function........ 191
GetNumInBytes Function........ 192
Read Function........ 193
Write Function........ 194
WriteString Function........ 195

PurgeComm Function... 196
Close Function... 196

rttlib.scheduler Module... 198
Parallel Generator Function... 198
ParallelRace Generator Function... 199

rttlib.utilities Module... 202
GetSequenceArgument Function.. 202
currentTime Variable Object... 203
Logging... 204
modelStepSize Variable Object.. 206
SequenceProperties.. 206
SetCallGCAfterRemovingAllSequences Function...................................... 208
SetImportTimeslice Function... 209
Wait Function... 211

rttlib.variable Module... 212
Variable Class.. 212
IsA2L Method... 214
Name Method... 215

rttlib.watcherlib Module.. 216
Watcher Class.. 216
Operators and Functions Supported by the watcherlib.............................. 218
Watch Method.. 221

Standard Python Libraries 223

Supported Python Modules.. 223
Supported Methods of the time Module.. 225

dSPACE.Common.MessageHandler.Logging Reference 227

ILogMessage Interface... 227
ILogSession Interface... 229
MessageReader Class... 230
MessageReaderSettings Class.. 232
Severity Enumeration... 233

Index 235

About This Reference

Contents

This reference introduces you to the standard and dSPACE Python modules for Real-Time Testing and the commands of the Real-Time Test Manager.

Required knowledge

Knowledge in handling the host PC and the Microsoft Windows operating system is assumed. This document is primarily targeted at engineers who have experience with the Python programming language.

Documented product versions

This documentation is part of several product versions of Real-Time Testing. As long as it is not stated, the descriptions are valid for all product versions. If there are differences, the product versions are stated.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
⍰	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.

Symbol	Description
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the 📄 icon in dSPACE Help. The PDF opens on the first page.

Real-Time Test Manager Commands

Introduction

The following topics describe the commands of the Real-Time Test Manager that is the graphical user interface for the RTT sequence control. The commands are available in the main menu and the context menus of the platform view and sequence list.

Where to go from here

Information in this section

Basic Interface...14

Platform Managing...21

Sequence Managing...31

Tools..39

Information in other sections

Managing RTT Sequences Using the Real-Time Test Manager (Real-Time Testing Guide 📖)
Describes how you can handle the RTT sequences on the real-time platform using the Real-Time Test Manager. This is the easiest way, but you cannot use all the features for handling RTT sequences.

Basic Interface

Where to go from here

Information in this section

About Real-Time Testing ... 14
To display information on your current Real-Time Test Manager version.

dSPACE Help ... 15
To open the user documentation of the Real-Time Test Manager.

Controlbars ... 15
To show or hide the Real-Time Test Manager's controlbars.

Exit ... 16
To exit the Real-Time Test Manager.

Log Viewer ... 16
To show or hide the Log Viewer.

New Features and Migration ... 18
To display new features and required migration steps for all the products in the current dSPACE Release.

Reset to Default ... 19
To reset the user interface settings to the default.

Status Bar ... 19
To show or hide the status bar at the bottom of the Real-Time Test Manager's main window.

Using dSPACE Help ... 20
To get information on working with dSPACE Help.

About Real-Time Testing

Access

To access this command via:

Menu bar	Help
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To display information on your current Real-Time Test Manager version.

Real-Time Test Manager dialog

Displays information on your current Real-Time Test Manager version.

dSPACE Help

Access

You can access this command via:

Menu bar	Help
Context menu of	None
Shortcut key	**F1**
Icon	None

Purpose

To open the user documentation of the Real-Time Test Manager.

Result

The user documentation of the Real-Time Test Manager opens.

Related topics

References

Using dSPACE Help..20

Controlbars

Access

You can access this command via:

Menu bar	View
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To show or hide the Real-Time Test Manager's controlbars.

Result

The Real-Time Test Manager's controlbars are either shown or hidden.

Description

Opens a submenu showing the controlbars available in the Real-Time Test Manager:

- Log
- Platform

Related topics

References

Reset to Default 19
Status Bar 19

Exit

Access

You can access this command via:

Menu bar	File
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To exit the Real-Time Test Manager.

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 🕮)

Log Viewer

Access

You can access this command via:

Menu bar	View – Controlbars – Log
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To show or hide the Log Viewer.

Result

The Log Viewer opens.

> **Tip**
>
> If the Log Viewer is open but not visible, you have to click the Log tab on the control bar.

Description

The Log Viewer provides a history of all error and warning messages that occur when you work with the Real-Time Test Manager. This helps you check the system state.

Severity, module, time, and text of a message Each message consists of several parts:

Part	Description
Severity	There are three types of messages according to severity level. Each message has a symbol that indicates the message type: ▪ ✖ Errors ▪ ⚠ Warnings ▪ ℹ Infos
Module	Module that the message comes from
Board	Board that the message comes from
Sequence	RTT sequence that the message comes from. It is empty if the message comes from the board.
Time	The time when the message occurred
Message	The content of the message

Buttons and commands of the Log Viewer

The Log Viewer provides several buttons and commands.

Copy (available from the context menu of messages) Lets you copy the complete entry of the message to the Clipboard.

Copy Message Text (available from the context menu of messages) Lets you copy the message text to the Clipboard.

Clear (available from the context menu of messages) Lets you clear all the messages in the Log Viewer.

Fix Scrolling (available from the context menu of messages) Lets you disable the automatic horizontal scrolling mechanism in the Log Viewer.

Sort Ascending (also available from the context menu of column headers) Lets you sort the grid alphabetically in ascending order according to the selected column.

Sort Descending (also available from the context menu of column headers) Lets you sort the grid alphabetically in descending order according to the selected column.

Column Chooser / View Column Chooser (available from the context menu of column headers) Lets you add a column to the grid and opens a dialog displaying the columns that can be added to the grid. To add a column, drag it from the dialog to the grid header. To remove a column from the grid, drag its header below the grid.

Best Fit (available from the context menu of column headers) Lets you optimize the width of the selected column.

Best Fit (all columns) (available from the context menu of column headers) Lets you optimize the widths of all columns to fit the width of the editor or browser.

New Features and Migration

Access

You can access this command via:

Menu bar	Help
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To display new features and required migration steps for all the products in the current dSPACE Release.

Result

dSPACE Help opens with New Features and Migration 🕮 displayed. Navigate to the specific product information to read about the new features of a specific product. If there are migration steps required, the necessary steps are described.

Reset to Default

Access

You can access this command via:

Menu bar	View
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To set the arrangement of the user interface to the original settings as and when the Real-Time Test Manager was first installed.

Result

The settings of the user interface are reset to the default.

Description

The controlbars of the Real-Time Test Manager, such as **Platform**, **Message Viewer**, and all other controlbars and toolbars, can be arranged according to your needs. For example, you can move a controlbar and dock it to another controlbar. All your modifications are saved when you exit the Real-Time Test Manager. You can use this command to reset the user interface to its default.

Related topics

References

Controlbars.. 15
Status Bar.. 19

Status Bar

Access

You can access this command via:

Menu bar	Views
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To show or hide the status bar at the bottom of the Real-Time Test Manager's main window.

Related topics

References

Controlbars.. 15
Reset to Default... 19

Using dSPACE Help

Access

You can access this command via:

Menu bar	Help
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To get information on working with dSPACE Help.

Result

dSPACE Help opens. It provides information on general handling and instructions on using dSPACE Help.

Platform Managing

Where to go from here

Information in this section

Connect....21
To access a real-time platform.

Create Sequence....22
To create an RTT sequence on the real-time platform.

Manage Recent Platform Configuration....23
To display and manage the simulation platforms that were registered in your system.

Refresh Platform Configuration....25
To refresh the platform connection from the host PC to the platform (for Real-Time Testing 2.1 and later).

Register Platforms....26
To register a platform.

Connect

Access

You can access this command via:

Menu bar	None
Context menu of	Platform name or application name in the **Platform** view
Shortcut key	None
Icon	None

Purpose

To access the currently selected simulation platform.

Note

This can take some time, for example, when accessing the simulation platform for the first time.

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 📖)

References

AccessBoard Method..49

Create Sequence

Access

You can access this command via:

Menu bar	None
Context menu of	Platform name or application name in the **Platform** view
Shortcut key	None
Icon	None

Purpose

To create an RTT sequence on the simulation platform.

> **Tip**
>
> You can create a new RTT sequence with default settings by dragging & dropping the RTT sequence file in PY or BCG file format onto the simulation platform.

Real-Time Test Manager Create Dialog

Real-Time Test Sequence Lets you specify the RTT sequence to be executed.

You can click ▾ to select an RTT sequence that was created before.

You can click the Browse button to select the RTT sequence in BCG or PY file format via the standard Open dialog.

Ignore missing modules Indicates whether imported modules are ignored when they are missing or an error message is given.

Sequence Channel Lets you select the point in time when the RTT sequence is executed, in relation to the model simulation:

- scPreComputation: In a sampling step, the RTT sequence is executed before the model simulation is executed by the real-time application.
- scPostComputation: In a sampling step, the RTT sequence is executed after the model simulation is executed by the real-time application.

Priority Lets you specify the RTT sequence's priority in a range from 1 to 256 with 1 as the highest priority. The sequences are executed in an order according to the specified priorities. If RTT sequences have the same priority, they are executed in the reverse order in which they are downloaded to the real-time platform. In a sampling step, the most recently created RTT sequence is then executed before older RTT sequences.

Description Lets you specify a user-defined description of the RTT sequence to be shown in the **Sequence** list of the Real-Time Test Manager.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

HowTos

How to Create a New RTT Sequence on the Platform (Real-Time Testing Guide 🕮)

References

Create Method.. 57

Manage Recent Platform Configuration

Access

You can access this command via:

Menu bar	**Tools**
Context menu of	**Platform** view
Shortcut key	None
Icon	None

Purpose

To display and manage the simulation platforms that were registered in your system.

Result

The Real-Time Test Manager opens the Manage Recent Platform Configuration dialog**Manage Recent Platform Configuration** dialog, which lets you manage your recent platform configuration. You can remove elements from the recent platform configuration and hide registered platforms in the **Platform** view. You can import configurations for registered platforms from an XML file or export the recent hardware configuration to an XML file.

Manage Recent Platform Configuration dialog

To manage the registered platforms and import/export the configuration of registered hardware.

Platforms Lists the simulation platforms that were registered in your system and whose registration data is stored in the recent platform configuration, and displays information on the registered platforms. The Platform list also provides an Active checkbox for each platform. If the checkbox is cleared, the platform is hidden and not displayed in the Platform view. If the checkbox is selected, the platform is listed and displayed in the Platform view.

Active Lets you specify to display the registered platform in the Platform view. If the checkbox is cleared, the platform is hidden and not displayed in the Platform view.

Remove To remove the currently selected platform from the recent hardware configuration. The platform is no longer available as a registered platform and is no longer displayed in the Platform view.

Remove All To remove all listed platforms from the recent hardware configuration. The platforms are no longer available as registered platforms and are no longer displayed in the Platform view.

Import Lets you select the XML file containing the platform configuration you want to import. The currently active platform configuration is replaced by the content of the imported XML file.

Export Lets you select the XML file you want to export to.

Commands The following commands are available from the context menus:

Command	Purpose
Best fit[1)]	To optimize the width of the selected column.
Best fit (all columns)[1)]	To optimize the widths of all columns according to the width of the editor or browser.
Collapse All[2)]	To collapse all the items and their subnodes in the platform list. ConfigurationDesk displays a reduced platform list.
Column Chooser[1)]	To open a dialog for customizing the columns of the recent hardware configuration grid. To add a column to the grid, drag it from the opened dialog to the grid header. To remove a column from the grid, drag its header to the dialog.
Expand All[2)]	To expand all the items and their subnodes in the platform list. ConfigurationDesk displays a detailed platform list.
Expand Default[2)]	To expand only the first-level items in the platform list. ConfigurationDesk displays a detailed platform list of the first-level items, but their subnodes are hidden.
Remove[2)]	To remove the currently selected registered platform from the recent hardware configuration. The platform is no

Command	Purpose
	longer available as a registered platform and is no longer displayed in the **Platform** view.
Remove All[2]	To remove all the registered platforms from the recent hardware configuration. The platforms are no longer available as registered platforms and are no longer displayed in the **Platform** view.
Select/Unselect All[2]	To select or clear the **Active** checkboxes of all the platforms in the platforms list. This lets you hide or show all registered platforms in the **Platform** view in one step.
Sort Ascending[1]	To sort the grid alphabetically in ascending order according to the selected column.
Sort Descending[1]	To sort the grid alphabetically in descending order according to the selected column.

[1] Available from the context menu of column headers

[2] Available from the context menu of platforms

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 📖)

Refresh Platform Configuration

Access

You can access this command via:

Menu bar	**Tools**
Context menu of	**Platform** view
Shortcut key	None
Icon	None

Purpose

To refresh the platform connection from the host PC to the platform (for Real-Time Testing 2.1 and later).

Result

The platform configurations are refreshed. The view of the structure shown in the **Platform** view is updated.

Description

The Real-Time Test Manager searches for registered platforms which are not displayed in the **Platform** view yet, and adds them to the **Platform** view. It also scans the recent hardware configuration for hardware that is not yet registered and tries to register it.

Use this command to start the search for the platforms manually. The search can also be started automatically when the Real-Time Test Manager starts, but the startup process can be affected by long timeouts. To enable the search during startup, select an option on the **Platform Management** page of the **General Properties** dialog.

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 🕮)

References

General Properties.. 40

Register Platforms

Access

You can access this command via:

Menu bar	**Tools**
Context menu of	Platform name in the **Platform** view
Shortcut key	None
Icon	None

Purpose

To register a platform.

Result

The Real-Time Test Manager now recognizes the registered platform.

Description

In the Real-Time Test Manager, the registered hardware systems are treated as platforms which are displayed and which can be accessed via the **Platform** view. To register platforms, you must enter their connection settings in the **Register Platforms** dialog.

When you click **Register** to register a new platform, the Real-Time Test Manager starts to search for the real-time hardware. If it is found, the platform is automatically read out and displayed in the dialog and in the **Platform** view.

> **Note**
>
> Your host PC must be connected to the same network as the hardware system you want to register in the Real-Time Test Manager.
>
> Using the MAC address, alias name, or board name to find and register the hardware is supported only if the host PC and hardware are part of the same subnetwork. If your hardware system is installed in a different subnetwork connected to your host PC's network via a router or gateway, you must use the IP address for registering. Otherwise the Real-Time Test Manager is unable to find the hardware system.

Register Platforms dialog

To specify the register settings for a single processor or controller board, multiprocessor system, MicroAutoBox II, DS6001 Processor Board, or SCALEXIO Processing Unit, and to get information on the platforms registered so far.

Platforms Lets you select the platform type being registered.

Platform properties Lets you view and specify the register settings for the platform. The following table shows all the possible properties. Only the properties relevant for the selected platform type are displayed:

Property	Description
Common Properties	
Platform name	Lets you specify a name for the selected DS6001 Processor Board, SCALEXIO Processing Unit or DS1007 platform. After registration, the name is displayed in the **Platform** view. If you do not specify a name for a SCALEXIO Processing Unit, "SCALEXIO Real-Time PC" is displayed in the **Platform** view.
Multiprocessor type	Displays the processor board type the multiprocessor system is based on.
Platform type	Displays the type of the selected platform.
Topology check	Lets you specify to check the topology of the selected multiprocessor system. If enabled, the Real-Time Test Manager checks whether all the processor boards of the system are interconnected via Gigalinks. The Real-Time Test Manager does not check whether the topology of the connected boards is compatible with the topology required by the real-time application to be loaded to the system, i.e., it does not check whether the correct Gigalink ports of the processor boards are used for interconnection. The Real-Time Test Manager performs the check when the multiprocessor system is connected.
Connection Settings Properties	
Alias name	Lets you specify the alias name of the connection that is used for assignment.

Property	Description
Connection parameter	Lets you select which connection parameter is listed under Processing unit or Processor board.
Connection type	Lets you specify the connection type of the platform hardware. ▪ Select "BUS" if the platform hardware is installed in the host PC or in an expansion box connected to the host PC via a bus interface. ▪ Select "NET" if the platform hardware is connected to the host PC via Ethernet. If you register a multiprocessor system, the connection type is specified for the multiprocessor system, so it is valid for all the processor boards belonging to the multiprocessor system.
Network client	Lets you specify the IP address if the connection type is "NET". If you register a multiprocessor system, the network client is specified for the multiprocessor system, so it is valid for all the processor boards belonging to the multiprocessor system.
Port address	Lets you specify the base address of the board as specified with the DIP switches or the rotary switches on the board.
Processing unit or Processor board	Lists all the processing units or processor boards that are found when the network is scanned.
Scan for available processing units or processor boards	To scan the local network for connected processing units or processor boards and select one or more units or boards to register. This opens the Scan Local Network for Processor Boards or Scan Local Network for Processing Units dialog, see below.
Multiprocessor Configuration Properties	
Processors	Lets you specify the number of processors belonging to the multiprocessor system. Click to add a processor, or click ☒ to delete the selected processor.
Processor name	Displays or lets you specify the name of the selected processor board.
Port address	Lets you specify the base address of the board as specified with the DIP switches or the rotary switches on the board.

Register Lets you complete the registration. The registered platform is displayed together with the platform properties in the Registered platforms list. The registered platform is also displayed in the Platform view.

Registered platforms list Displays all the registered platforms with the following information: platform name, platform type, serial number/identifier, MAC address, network client, and port address.

You can customize the display in the Registered platforms list using the following commands available from the context menu of column headers:

- Best Fit: Lets you optimize the width of the selected column.
- Best Fit (all columns): Lets you optimize the widths of all columns according to the width of the editor or browser.
- Column Chooser: Lets you open a dialog for customizing the columns of the platforms list. To add a column to the list, drag it from the opened dialog to the list header. To remove a column from the list, drag its header to the dialog.

- Sort Ascending: Lets you sort the list alphabetically in ascending order according to the selected column.
- Sort Descending: Lets you sort the list alphabetically in descending order according to the selected column.

Scan Local Network for Processor Boards/ Processing Units dialog

To scan the local network for connected platform hardware or simulators, and select one or more platforms or a simulator to register.

Type Lets you select the filter item type you want to use to filter the results list. If you select 'None', no filtering is applied.

Value Lets you enter a filter string.

Match whole word Lets you specify to search only for a matching pattern substring.

(Re)scan Lets you start a new scan process. The Real-Time Test Manager scans the subnetwork your host PC is connected to for connected processor boards/processing units matching the specified filter settings, and refreshes the results list.

List of available processor boards/processing units Displays all the processor boards or processing units that the specified filter found in the network during the scan process. The results list contains the IP address, MAC address, board name, system name and serial number for each processor boards or processor board, processing unit that was found. If the scan process is performed for VEOS, the results list contains the IP address and host name for each found simulator, together with the respective product version and installation path of the VEOS installation on the simulator.

To select a processor board or processing unit for registration, click its entry and then press the ↓ button. The selected element is moved to the list of selected processor boards/processing units, where you can transfer its connection parameter value to the Register Platforms dialog.

Tip

You can multiselect processing units and processor boards.

List of selected processor boards/processing units Displays all the processor boards or processing units selected for registration so far. When you click Apply, the listed platform hardware is assigned to the platform you want to register, and the connection parameter value of each list item is transferred to the Register Platforms dialog.

The following buttons are available to move elements from one list to the other:

↓	Moves the selected element(s) from the Available processor boards/processing units list to the Selected processor boards/processing units list.

⬆	Moves the selected element(s) from the Selected processor boards/processing units list to the Available processor boards/processing units list.

Apply Lets you confirm the selection of processor board(s) or processing unit(s) for registration. When you click this button, the connection parameter value of each element in the Selected processor boards or Selected processing units list is stored in the Register Platforms dialog.

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 📖)

References

Manage Recent Platform Configuration.. 23
Refresh Platform Configuration.. 25

Sequence Managing

Where to go from here

Information in this section

Continue.....31
To continue the RTT sequence execution at the point where it was paused.

Copy Error Text.....32
To copy the text of an error message if an RTT sequence has an error state.

Delete.....32
To remove an RTT sequence from the simulation platform.

Open.....33
To open the Python file of the selected RTT sequence in the standard program for Python files.

Pause.....34
To pause a running RTT sequence.

Reload.....34
To reload an RTT sequence from the host PC to the simulation platform.

Run.....35
To start an RTT sequence on the simulation platform.

Sequence List.....36
To display the RTT sequences created on the simulation platforms.

Stop.....37
To stop a running RTT sequence.

Continue

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	None
Icon	None

Purpose

To continue the RTT sequence execution at the point where it was paused.

Description

You can also continue the RTT sequence execution for all paused RTT sequences on a simulation platform in one step using multiselection.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 🕮)

References

Continue Method.. 70

Copy Error Text

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	`Ctrl` + `C`
Icon	None

Purpose

To copy the text of an error message if an RTT sequence has the error state `rttmanager.constants.sesError`.

Delete

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	`Del`
Icon	None

Purpose

To remove an RTT sequence from the simulation platform.

Description

When you remove an RTT sequence from the simulation platform, it is also removed from the sequence list of the Real-Time Test Manager.

To create an RTT sequence again, use **Create Sequence**.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

References

Create Sequence.. 22
Remove Method.. 60

Open

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	None
Icon	None

Purpose

To open the Python file of the selected RTT sequence in the standard program for Python files, for example, PythonWin.

Related topics

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 🕮)

Pause

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	None
Icon	None

Purpose

To pause a running RTT sequence.

Description

The running RTT sequence is paused but not stopped.

To continue the sequence execution at the point where it was paused, select **Continue** from the context menu. To restart the RTT sequence from the beginning, select **Run**.

You can pause all RTT sequences of a simulation platform in one step using multiselection.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 📖)

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 📖)

References

Continue.. 31

Reload

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list

Shortcut key	None
Icon	None

Purpose

To reload an RTT sequence from the host PC to the simulation platform.

Description

When you reload an RTT sequence, it is removed from the simulation platform and the Real-Time Test Manager. Afterwards, the RTT sequence is reloaded to the simulation platform and displayed in the Real-Time Test Manager's sequence list with the New state.

Related topics

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 📖)

Run

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	None
Icon	None

Purpose

To start an RTT sequence on the simulation platform.

Description

When the Real-Time Test Manager starts an RTT sequence, the sequence is executed on the simulation platform. An RTT sequence can be started if it has one of the following states:

- New
- Paused
- Stopped
- Terminated

When an RTT sequence is started with the Paused, Stopped, or Terminated state, its namespace is maintained. The sequence is not initialized but starts directly with executing the MainGenerator function.

The RTT sequences are executed according to their priorities. Refer to Create Sequence on page 22.

You can start all RTT sequences of a simulation platform which have the New state in one step using multiselection.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 🕮)

Sequence List

Purpose

To display the RTT sequences created on the simulation platforms.

Description

The Sequence list shows a table containing all RTT sequences created on the simulation platforms. It also displays the files that are used for data streaming and global variables.

Context menu of the column header

The column header has a context menu with the following commands.

Best Fit Lets you optimize the width of the selected column.

Best Fit (all columns) Lets you optimize the widths of all columns according to the width of the editor or browser.

Clear Sorting Lets you deactivate the sort attribute applied to the selected column.

Column Chooser Lets you add a column to the grid and opens a dialog displaying the columns that can be added to the grid. To add a column, drag it from the dialog to the grid header. To remove a column from the grid, drag its header below the grid.

Filter Editor Lets you open the Filter Editor to specify a filter for the grid. For an instruction on how to define a filter, refer to How to Specify and Use a Filter (Real-Time Testing Guide 📖).

Remove This Column Lets you remove the selected column.

Show Auto Filter Row Lets you show the Auto Filter Row.

Sort Ascending Lets you sort the grid alphabetically in ascending order according to the selected column.

Sort Descending Lets you sort the grid alphabetically in descending order according to the selected column.

Related topics

HowTos

How to Create a New RTT Sequence on the Platform (Real-Time Testing Guide 📖)
How to Customize the Screen Arrangement (Real-Time Testing Guide 📖)

Stop

Access

You can access this command via:

Menu bar	None
Context menu of	Sequence in the sequence list
Shortcut key	None
Icon	None

Purpose

To stop a running RTT sequence.

Description

When you stop a running RTT sequence, it is stopped but it remains on the simulation platform.

You can stop all RTT sequences of a simulation platform in one step using multiselection.

To remove the RTT sequences from the simulation platform, you must delete each of them individually. Refer to Delete on page 32.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

HowTos

How to Manage RTT Sequences on the Real-Time Platform (Real-Time Testing Guide 🕮)

Tools

Where to go from here

Information in this section

Explore Demos Folder..39
To explore the folder with the demo files.

General Properties...40
To specify additional module paths and settings for the view and platform management.

Unzip Demos...41
To copy the demo files to your working folder.

Explore Demos Folder

Access

You can access this command via:

Menu bar	Tools
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To explore the folder with the demo files.

Description

The Real-Time Test Manager opens Windows Explorer and browses to the demos folder.

Related topics

Examples

Demo Examples of Using Real-Time Testing (Real-Time Testing Guide 🕮)

General Properties

Access

You can access this command via:

Menu bar	Tools
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To specify additional module paths and settings for the view and platform management.

General page

The General page lets you specify additional module paths.

To add a new module path, enter the path in the edit box or click …. A dialog opens where you can browse to the new folder.

To delete folders from the list, select the folders and click ×.

Platform Management

The Platform Management page lets you specify settings for the platform management.

Seek connected platforms on startup Lets you specify whether to search for registered platforms when the Real-Time Test Manager is started.

- If the checkbox is selected, the Real-Time Test Manager scans the recent hardware configuration and searches
  - For registered platforms connected via bus interface
  - For connected platforms that do not need to be registered (MicroAutoBox connected via bus)
  - For registered and connected SCALEXIO systems

  The platforms that are found are displayed in the Platform view.
- If the checkbox is cleared, the Real-Time Test Manager does not search for connected and registered platforms during startup.

Seek MicroAutoBox II and platforms connected via slot CPU, too (Available only if Seek connected platforms on startup is selected) Lets you specify whether the Real-Time Test Manager should also search for registered MicroAutoBox IIs and registered platforms connected via slot CPU during startup.

Note

If this option is enabled, the startup process can be affected by long timeouts.

If this option is disabled, you can start the platform search manually, see Refresh Platform Configuration on page 25.

View

The View page lets you specify view settings.

Save view settings on close Lets you enable saving the view settings when you exit the Real-Time Test Manager.

Related topics

HowTos

How to Start the Real-Time Test Manager and Access a Platform (Real-Time Testing Guide 📖)

Unzip Demos

Access

You can access this command via:

Menu bar	Tools
Context menu of	None
Shortcut key	None
Icon	None

Purpose

To copy the demo files to your working folder.

Description

When the demo files are unpacked to your documents folder, you can examine several short examples of an RTT sequence. These are ready-to-use RTT sequences with the TurnSignal demo.

The ControlDesk projects for all systems are installed in `SampleExperiments\TurnSignal_<platform>` (<platform> is an abbreviation of the used platform).

Related topics

Examples

Demo Examples of Using Real-Time Testing (Real-Time Testing Guide 📖)

dSPACE Python Modules for Managing RTT Sequences

Introduction

These modules are available on the host PC. You can use them to manage the RTT sequences and the Real-Time Test Manager Server.

Where to go from here

Information in this section

rttmanagerlib Module.....44
The Real-Time Test Manager Server handles the RTT sequences on the host PC and creates them on the simulation platform.

rttutilities Module.....86
Provides functions for on the host PC that are useful for real-time testing but not available in the standard Python libraries.

rttmanagerlib Module

Introduction

The Real-Time Test Manager Server handles the RTT sequences on the host PC and creates them on the simulation platform.

Where to go from here

Information in this section

rttmanagerlib Module Quick Reference....................45

RealTimeTestManagerServer....................47
To handle objects of the Board type.

BCGServiceProvider....................52
To generate or sign a BCG file of an RTT sequence.

Board....................54
To access the specified real-time platform.

Sequences (Collection)....................55
To manage the collection of RTT sequences on a real-time platform.

SequencesEvents....................62
To handle events of the RTT sequences.

Sequence....................68
To manage an RTT sequence.

SequenceEvents....................74
To handle events of an RTT sequence.

DataStreams....................77
To get information on the datastreams which are used in an RTT sequence.

ExecutionError....................78
To get information on last errors occurring during sequence execution.

Variables (Collection)....................79
To manage the collection of dynamic variables on a real-time platform.

VariablesEvents....................80
To handle events of the dynamic variables.

Variable....................83
To manage a dynamic variable.

Common Constants....................84
To specify the common attributes of rttmanagerlib.

rttmanagerlib Module Quick Reference

Introduction

Object information of the rttmanagerlib module is summarized in a compact table, which provides a quick overview of the available objects, object dependencies, attributes and methods.

Overview of the rttmanagerlib Object Model

Introduction

The object model overview of the rttmanagerlib module gives a quick overview of object dependencies, and available object attributes and methods.

Symbols

The following symbols are used in the object model overview:

Symbol	Description
◆	Method, function
▤	Attribute (property, class)
▦	Collection
0, 1, 2, …	Level of dependency (0, 1, 2, …)
⊗	Read only

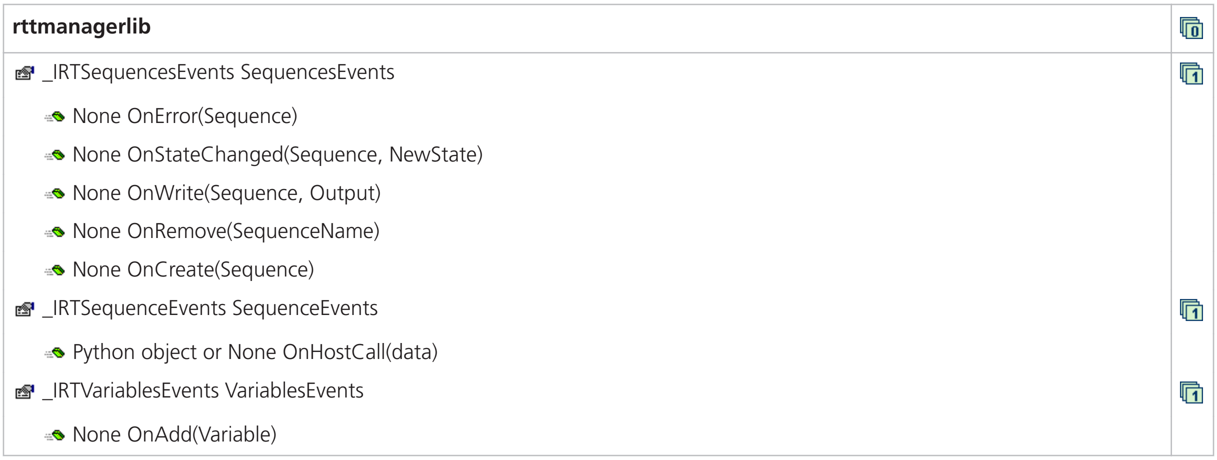
rttmanagerlib

The following table gives an overview of the rttmanagerlib's object model:

rttmanagerlib	0
▤ rttmanagerlib constants	
▤ RealTimeTestManagerServer RealTimeTestManagerServer	1
▤ string Version	
▤ string PythonToolsPath	
▤ BCGServiceProvider BCGServiceProvider	
◆ BCGFileName Generate(FileName, UserSearchPath, IgnoreMissingModules = False, Sign = True)	2
◆ string ApplicationRTTVersion(BoardName)	1
◆ Board AccessBoard(BoardName)	
▦ Boards Boards	2
▤ int Count	
▤ Board item(index)	3
▤ string Name	

rttmanagerlib

- Sequences Sequences
  - int Count
  - None ContinueAll()
  - Sequence Create(FileName, Data = " ", SequenceChannel = rttmanagerlib.constants.scPreComputation, Priority = 1, Option = 0, Description = " ")
  - None PauseAll()
  - None Remove(Index)
  - None RunAll()
  - None StopAll()
  - Sequence Item(Index)
    - string Description
    - string FileName
    - int Handle
    - ExecutionError LastExecutionError
    - string Name
    - int Priority
    - int SequenceChannel
    - int State
    - ULong64 ActiveTime
    - ULong64 RunningTime
    - ULong64 StepSize
    - None Continue()
    - None Pause()
    - None Remove()
    - None Run(RunParameter)
    - None Stop()
    - DataStreams DataStreams
- Variables Variables
  - int Count
  - Variable Item(Index)
    - string Name
    - string SequenceName
    - float Value
    - float DataType
    - string Description
    - string PathName
    - float DynamicValue



rttmanagerlib

_IRTSequencesEvents SequencesEvents
- None OnError(Sequence)
- None OnStateChanged(Sequence, NewState)
- None OnWrite(Sequence, Output)
- None OnRemove(SequenceName)
- None OnCreate(Sequence)

_IRTSequenceEvents SequenceEvents
- Python object or None OnHostCall(data)

_IRTVariablesEvents VariablesEvents
- None OnAdd(Variable)

Related topics

Basics

Managing RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

References

rttmanagerlib Module..44

RealTimeTestManagerServer

Where to go from here

Information in this section

RealTimeTestManagerServer Class Description..48
To handle objects of the Board type.

AccessBoard Method..49
To access a simulation platform for real-time testing.

ApplicationRTTVersion Method...50
To get the version of Real-Time Testing that is executed with a simulation application.

Information in other sections

Board Class Description..54
To access the specified real-time platform.

RealTimeTestManagerServer Class Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("ds1006")
```

Purpose

To handle objects of the Board type.

> **Note**
>
> The Real-Time Test Manager Server shuts down as soon as the Real-Time Test Manager object is terminated. The Real-Time Test Manager object must therefore be available as long as other objects are in use.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
PythonToolsPath	String	To get the absolute path to the `Tools` folder of the active installation of Real-Time Testing.
Version	String	To get the version of Real-Time Testing that is active on the host PC.

Methods

The following method is part of the class:

Method	Purpose
AccessBoard	To access a simulation platform. Refer to AccessBoard Method on page 49
ApplicationRTTVersion	To get the version of Real-Time Testing that is executed on a real-time platform. Refer to ApplicationRTTVersion Method on page 50.

Example

The following example shows how to get information on Real-Time Testing.

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
RTTVersion = rttm.ApplicationRTTVersion("127.0.0.1")
print("RTT platform version", RTTVersion)
print("RTT host PC version: ", rttm.Version)
print("RTT Python tools path: ", rttm.PythonToolsPath)
```

Related topics

References

Board Class Description.. 54

AccessBoard Method

Class

RealTimeTestManagerServer

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard(BoardName)
```

Purpose

To access a simulation platform for real-time testing.

Description

You can use the method to access single-processor and multiprocessor systems registered on the host PC. For details, refer to Creating and Starting RTT Sequences in Python Scripts (Real-Time Testing Guide 📖).

Parameter

The method uses the following parameter:

Parameter	Type	Description
BoardName	String	Name or IP address of the simulation platform. The notation is not case-sensitive. The usage depends on the platform type: ▪ For a single processor board (DS1006, MicroAutoBox), BoardName is the platform name. The platform name is specified when the board is registered. It is displayed in the Platforms/Devices controlbar of ControlDesk, for example. This is independent of the connection type. ▪ For a multiprocessor system (DS1006), you must access each processor board individually using the platform name. For example, if ControlDesk displays the platform names `ds1006_2` and `ds1006_3` in the Platforms/Devices controlbar, use

Parameter	Type	Description
		`AccessBoard("ds1006_2")` and afterwards `AccessBoard("ds1006_3")`. ▪ For a DS1007, MicroLabBox, DS6001, and SCALEXIO, BoardName is the IP address of the platform and the application name separated by a slash, for example, `AccessBoard("192.168.0.15/MyApp")`. ▪ For VEOS, BoardName is the IP address of the host PC where VEOS runs and the application name separated by a slash, for example, `AccessBoard("127.0.0.1/MyApp")` if VEOS runs on the same PC where Real-Time Testing is installed.

Return value

The method returns a value of the following type:

Type	Description
Board	To access the specified simulation platform.

Related topics

References

Board Class Description..54
RealTimeTestManagerServer Class Description..48

ApplicationRTTVersion Method

Class

RealTimeTestManagerServer

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Version = rttm.ApplicationRTTVersion(BoardName)
```

Purpose

To get the version of Real-Time Testing that is executed with a simulation application.

Description

The version of Real-Time Testing that is executed with the real-time application must be equal to the version of Real-Time Testing that is executed on the host PC.

You can use the method to read the version of Real-Time Testing on the platform. The version that is executed on the host PC is specified by the `Version` attribute of a RealTimeTestManagerServer object.

If Real-Time Testing is not enabled for the simulation application or a simulation application is not executed, an exception is thrown.

Parameter

The method uses the following parameter:

Parameter	Type	Description
BoardName	String	Name or IP address of the simulation platform. The notation is not case-sensitive. The usage depends on the platform type: ▪ For a single processor board (DS1006, MicroAutoBox), BoardName is the platform name. The platform name is specified when the board is registered. It is displayed in the **Platforms/Devices** controlbar of ControlDesk, for example. This is independent of the connection type. ▪ For a multiprocessor system (DS1006), you must access each processor board individually using the platform name. For example, if ControlDesk displays the platform names `ds1006_2` and `ds1006_3` in the **Platforms/Devices** controlbar, use `AccessBoard("ds1006_2")` and afterwards `AccessBoard("ds1006_3")`. ▪ For a DS1007, MicroLabBox, DS6001, and SCALEXIO, BoardName is the IP address of the platform and the application name separated by a slash, for example, `AccessBoard("192.168.0.15/MyApp")`. ▪ For VEOS, BoardName is the IP address of the host PC where VEOS runs and the application name separated by a slash, for example, `AccessBoard("127.0.0.1/MyApp")` if VEOS runs on the same PC where Real-Time Testing is installed.

Return value

The method returns a value of the following type:

Type	Description
String	The version of Real-Time Testing executed on the specified simulation platform.

Related topics

References

RealTimeTestManagerServer Class Description..48

BCGServiceProvider

Where to go from here

Information in this section

BCGServiceProvider Class Description....52
To generate and sign a BCG file of an RTT sequence.

Generate Method....53
To generate a BCG file of an RTT sequence.

BCGServiceProvider Class Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
MyBCGServiceProvider = rttm.BCGServiceProvider
```

Purpose

To generate and sign a BCG file of an RTT sequence.

Attributes

–

Methods

The following methods are part of the class:

Method	Purpose
Generate	To generate a BCG file of an RTT sequence. Refer to Generate Method on page 53.

Related topics

Basics

Creating and Starting RTT Sequences in Python Scripts (Real-Time Testing Guide 🕮)

Generate Method

Class

BCGServiceProvider

Syntax

```
rttm.BCGServiceProvider.Generate(FileName, UserSearchPath,
IgnoreMissingModules = False, Sign = True)
```

Purpose

To generate a BCG file of an RTT sequence.

Note

Signing an RTT sequence is only possible with the RTT_DEVELOPER license of Real-Time Testing.

Parameter

The function uses the following parameters:

Parameter	Type	Description
FileName	String	The Python file in which the RTT sequence is implemented.
UserSearchPath	List	A list of paths to folders including user modules which are imported into the Python file. Compiled modules in the PYC format are imported if they were compiled with the same Python version.
IgnoreMissingModules	Boolean	Indicates how to deal with missing modules from import statements (optional): ▪ True: Missing files are ignored. ▪ False: Error when files are missing. The default is False.
Sign	Boolean	Signs a generated BCG file (optional): ▪ True: The generated BCG file is signed. ▪ False: The generated BCG file is not signed. The default is True.

Return value

The function returns a value of the following type:

Type	Description
String	Returns the path to the generated BCG file.

Note

The generated BCG file is usually placed in the same folder as the source file. If the folder already contains a BCG file with the same name, that file is replaced if it is older than the source file of the new BCG file. If the file is read-only, an exception occurs.

Related topics

Basics

Creating and Starting RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

References

BCGServiceProvider Class Description.. 52

Board

Board Class Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
```

Purpose

To access the specified real-time platform.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
Name	String	To get the name of the simulation platform, for example, "127.0.0.1/MyApp". The notation is not case-sensitive.
PythonVersion	String	To get the version of the Python interpreter running on the platform. The string has the following form: `<Major>.<Minor>.<Maintenance>` For example, "3.6.4", or "2.7.11"
Sequences	Sequences[1)]	To get the collection of RTT sequences.

Attribute	Type	Purpose
Variables	Variables[2]	To get the collection of the dynamic variables.

[1] Refer to Sequences (Collection) on page 55.

[2] Refer to Variables (Collection) on page 79.

Methods

–

Related topics

References

Sequences (Collection)..55
Variables (Collection)...79

Sequences (Collection)

Where to go from here

Information in this section

Sequences Class (Collection) Description..56
To manage the collection of RTT sequences on a simulation platform.

ContinueAll Method...57
To continue the execution of all RTT sequences on a simulation platform.

Create Method...57
To create a new RTT sequence on the simulation platform.

Item Method..59
To return an RTT sequence by index.

PauseAll Method...60
To pause all the RTT sequences running in the same sampling step on the simulation platform.

Remove Method..60
To remove an item from the collection.

RunAll Method..61
To start all new RTT sequences on the simulation platform in the same sampling step.

StopAll Method..61
To stop all RTT sequences running in the same sampling step on the simulation platform.

Sequences Class (Collection) Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Board.Sequences
```

Purpose

To manage the collection of RTT sequences on a simulation platform.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Count	Integer	To get the number of items in the collection

Methods

The following methods are part of the class:

Method	Purpose
ContinueAll	To continue the execution of all RTT sequences on a simulation platform. Refer to ContinueAll Method on page 57.
Create	To download a new RTT sequence to the simulation platform. Refer to Create Method on page 57.
Item	To return an RTT sequence by index. Refer to Item Method on page 59.
PauseAll	To pause all running RTT sequences on a simulation platform. Refer to PauseAll Method on page 60.
Remove	To remove an item from the collection by index. Refer to Remove Method on page 60.
RunAll	To start all RTT sequences on the simulation platform. Refer to RunAll Method on page 61.
StopAll	To stop all running RTT sequences on a simulation platform. Refer to StopAll Method on page 61.

Related topics

Basics

Managing RTT Sequences in Python Scripts (Real-Time Testing Guide 🕮)

ContinueAll Method

Class

Sequences

Syntax

```
OBJ.ContinueAll()
```

Purpose

To continue the execution of all RTT sequences on a simulation platform.

Parameter

–

Return value

–

Related topics

References

Sequences Class (Collection) Description 56

Create Method

Class

Sequences

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Sequence = Board.Sequences.Create(FileName, pPickleData = "",\
   SequenceChannel = rttmanagerlib.constants.scPreComputation,\
   Priority = 1,Option = 0,Description = "")
```

Purpose

To create a new RTT sequence on the simulation platform.

Description

Each RTT sequence created on the simulation platform has its own namespace. To exchange values between different RTT sequences, you can use the globalvariables module, refer to rttlib.globalvariables Module on page 187.

Demo files

Some demo examples are installed with Real-Time Testing, refer to Demo Examples of Using Real-Time Testing (Real-Time Testing Guide 🕮).

Parameter

The method uses the following parameters:

Parameter	Type	Description
FileName	String	Name of the BCG file which is downloaded to the RTT sequence
pPickleData		Python object that is passed to the started RTT sequence with the `GetSequenceArgument` function.[1]
SequenceChannel	Integer	Time when the RTT sequence is executed: ▪ scPreComputation: The RTT sequence is executed before the simulation model is calculated by the real-time application. ▪ scPostComputation: The RTT sequence is executed after the simulation model is calculated by the real-time application. The parameter is optional. The default value is scPreComputation.
Priority	Integer	Priority of the RTT sequence in a range from 1 to 256 with 1 as the highest priority. The priority specifies the execution order of the RTT sequences. If RTT sequences have the same priority, they are executed in the reverse order in which they are created on the platform. In a sampling step, the most recently created RTT sequence is then executed before older RTT sequences. The parameter is optional. The default value is 1.
Option	-	Not supported with the current version of Real-Time Testing.
Description	String	User-defined description for the RTT sequence shown in the Real-Time Test Manager. The parameter is optional. The default value is `""`.

[1] Refer to GetSequenceArgument Function on page 202.

Return value

The method returns a value of the following type:

Type	Description
Sequence	To manage an RTT sequence.

If an error occurs and the RTT sequence is not created, an exception is raised.

Related topics

Basics

Basics on Executing RTT Sequences (Real-Time Testing Guide 📖)
Implementing an Exception Handling (Real-Time Testing Guide 📖)
States of RTT Sequences (Real-Time Testing Guide 📖)
Using Variables Accessible by Several RTT Sequences (Real-Time Testing Guide 📖)

References

ExecutionError Class Description.. 78
Sequence Class Description.. 69

Item Method

Class

Sequences

Syntax

```
RetVal = OBJ.Item(Index)
```

Purpose

To return an RTT sequence by index.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Item	Integer or String	Index or Name of the RTT sequence

Return value

The method returns a value of the following type:

Type	Description
Sequence	To manage an RTT sequence.

Related topics

References

Sequences Class (Collection) Description.. 56

PauseAll Method

Class

Sequences

Syntax

```
OBJ.PauseAll()
```

Purpose

To pause all the RTT sequences running in the same sampling step on the simulation platform.

Parameter

–

Return value

–

Related topics

References

Sequences Class (Collection) Description.. 56

Remove Method

Class

Sequences

Syntax

```
OBJ.Remove(Index)
```

Purpose

To remove an item from the collection.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Index	Integer	Index of the RTT sequence to be removed

Return value

–

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

RunAll Method

Class

Sequences

Syntax

```
OBJ.RunAll()
```

Purpose

To start all new RTT sequences on the simulation platform in the same sampling step.

Description

RTT sequences that were already executed and do not have the New state, are not started again.

Parameter

–

Return value

–

Related topics

References

Run Method... 72

StopAll Method

Class

Sequences

Syntax

```
OBJ.StopAll()
```

Purpose	To stop all RTT sequences running in the same sampling step on the simulation platform.
Parameter	–
Return value	–
Related topics	References Sequences Class (Collection) Description 56

SequencesEvents

Where to go from here

Information in this section

SequencesEvents Class Description 63
To handle events of the RTT sequences.

OnCreate Method 65
To output the RTT sequence object that was created.

OnError Method 65
To output the RTT sequence object in which the error occurred.

OnRemove Method 66
To output the name of the RTT sequence removed from the simulation platform.

OnStateChanged Method 67
To output the RTT sequence object in which the state changed.

OnWrite Method 67
To output the RTT sequence object.

SequencesEvents Class Description

Syntax

See example below.

Purpose

To handle events of the RTT sequences.

Description

RTT sequences can trigger events which you can evaluate in the host PC script. You can also implement event handling in the script. For an example, refer to Handling Events of RTT Sequences in Python Scripts (Real-Time Testing Guide 🕮).

Attributes

–

Methods

The following methods for the events are defined in the class:

Method	Description
OnError	An error occurred in the RTT sequence, specified by `Sequence`. Refer to OnError Method on page 65.
OnStateChanged	The state of the RTT sequence, specified by `Sequence`, changed to a new state, specified by `NewState`. Refer to OnStateChanged Method on page 67.
OnWrite	The print command, specified by `print`, prints the string `Output` in the RTT sequence, specified by `Sequence`. Refer to OnWrite Method on page 67.
OnRemove	The RTT sequence, specified by `SequenceName`, was removed from the simulation platform. Refer to OnRemove Method on page 66.
OnCreate	The RTT sequence, specified by `Sequence`, was created. Refer to OnCreate Method on page 65.

Tip

- If the `EventObject` is destroyed, no events are output.
- Use `rttutilities.RTTSleep()` to wait for events, for example, `OnWrite` events after executing `Sequence.Run()`. If you use other Sleep functions, for example, `win32api.Sleep()`, the events are not output and it may lead to a deadlock.

Example

```
import rttmanagerlib
```

```
class RTTMSequencesEvents(rttmanagerlib._IRTSequencesEvents):
    def __init__(self, EventSource):
        # Call base class constructor to connect to event source
        rttmanagerlib._IRTSequencesEvents.__init__(self, EventSource)
    def OnError(self, Sequence):
        """Method OnError"""
        Sequence = rttmanagerlib.Sequence(Sequence)
        print("OnError: ", Sequence.Name)
    def OnStateChanged(self, Sequence, NewState):
        """Method OnStateChanged"""
        Sequence = rttmanagerlib.Sequence(Sequence)
        print("OnStateChanged: ", Sequence.Name)
    def OnWrite(self, Sequence, Output):
        """Method OnWrite"""
        Sequence = rttmanagerlib.Sequence(Sequence)
        print("OnWrite: ", Sequence.Name)
    def OnRemove(self, Name):
        """Method OnRemove"""
        print("OnRemove: ", Name)
    def OnCreate(self, Sequence):
        """Method OnCreate"""
        Sequence = rttmanagerlib.Sequence(Sequence)
        print("OnCreate: ", Sequence.Name)
def main():
    SequencesEvents = None
    rttm = rttmanagerlib.RealTimeTestManagerServer()
    try:
        Board = rttm.AccessBoard("127.0.0.1/MyApp")
        # Connect to sequences event handle
        SequencesEvents = RTTMSequencesEvents(Board.Sequences)
        #...
    finally:
        if SequencesEvents:
            # Disconnect from sequences event handle
            SequencesEvents.close()
            SequencesEvents = None
        Board = None
        rttm = None
#-----------------------------------------------------------
# Module main block
#-----------------------------------------------------------
if __name__ == '__main__':
    main()
```

Related topics

Basics

Handling Events of RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

OnCreate Method

Class

SequencesEvents

Syntax

```
def OnCreate(self, Sequence):
    Sequence = rttmanagerlib.Sequence(Sequence)
```

Purpose

To output the RTT sequence object that was created.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Sequence	Sequence[1)]	The RTT sequence object that was created.

[1)] Refer to Sequence Class Description on page 69.

Return value

–

Related topics

References

SequencesEvents Class Description 63

OnError Method

Class

SequencesEvents

Syntax

```
def OnError(self,Sequence):
    Sequence = rttmanagerlib.Sequence(Sequence)
```

Purpose

To output the RTT sequence object in which the error occurred.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Sequence	Sequence[1]	The RTT sequence object in which the error occurred. Get `Sequence.LastExecutionError` to get the latest error.

[1] Refer to Sequence Class Description on page 69.

Return value

–

Related topics

References

SequencesEvents Class Description.. 63

OnRemove Method

Class

SequencesEvents

Syntax

```
OnRemove(SequenceName)
```

Purpose

To output the name of the RTT sequence removed from the simulation platform.

Parameter

The method uses the following parameter:

Parameter	Type	Description
SequenceName	String	The name of the removed RTT sequence.

Return value

–

Related topics

References

SequencesEvents Class Description.. 63

OnStateChanged Method

Class

SequencesEvents

Syntax

```
def OnStateChanged(self, Sequence, NewState)
   Sequence = rttmanagerlib.Sequence(Sequence)
```

Purpose

To output the RTT sequence object in which the state changed.

Parameter

The method uses the following parameters:

Parameter	Type	Description
Sequence	Sequence[1]	The sequence object in which the state changed to `NewState`.
NewState	Integer	The new state of the RTT sequence.

[1] Refer to Sequence Class Description on page 69.

Return value

–

Related topics

References

SequencesEvents Class Description.. 63

OnWrite Method

Class

SequencesEvents

Syntax

```
def OnWrite(self, Sequence, Output):
   Sequence = rttmanagerlib.Sequence(Sequence)
```

Purpose

To output the RTT sequence object.

Description

The method writes the output of the print command in an RTT sequence to the host PC.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Sequence	Sequence[1]	The RTT sequence object.
Output	String	The print output of the RTT sequence object

[1] Refer to Sequence Class Description on page 69.

Return value

–

Example

For example, if you use the following code in an RTT sequence:

```
print("RTT %s" % 42)
```

The string is output in the log window of the host PC:

```
RTT 42
```

Related topics

References

SequencesEvents Class Description.. 63

Sequence

Where to go from here

Information in this section

Sequence Class Description..69
To manage an RTT sequence.

Continue Method..70
To continue the sequence execution at the point where it was paused.

Pause Method...71
To pause a running RTT sequence.

Remove Method..71
To remove an RTT sequence from the real-time platform.

Run Method..72
To start an RTT sequence on the simulation platform.

Stop Method...73
To stop a running RTT sequence.

Sequence Class Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Sequence = Board.Sequences[0]
```

Purpose

To manage an RTT sequence.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Description	String	To get the RTT sequence description you specified when creating the RTT sequence. Refer to Create Method on page 57.
FileName	String	To get the absolute BCG file name of the RTT sequence.
Handle	Integer	To handle the RTT sequence on the simulation platform.
LastExecutionError	ExecutionError[1)]	To get information on errors occurring during sequence execution. If no error occurred, the return value is None.
Name	String	To get the name of the RTT sequence
Priority	Integer	To get the RTT sequence's position in the priority list in a range from 1 to 256 with 1 as the highest priority.
SequenceChannel	Integer	To get the sequence channel `rttmanagerlib.constants.scPreComputation` or `rttmanagerlib.constants.scPostComputation`
State	Integer	To get the state of the RTT sequence, for example, `constants.sesError`. Refer to Common Constants of rttmanagerlib on page 84.
Datastreams	DataStreams[2)]	To get information on the datastreams used in the RTT sequence.
ActiveTime	ULong64	To get the active time (running and paused time) of the RTT sequence in nanoseconds (10^{-9} s). To read the attribute, you must cast the `Sequence` object[3)].
RunningTime	ULong64	To get the running time (without pause) of the RTT sequence in nanoseconds (10^{-9} s). To read the attribute, you must cast the `Sequence` object[3)].

Attribute	Type	Purpose
StepSize	ULong64	To get the step size of the model in nanoseconds (10^{-9} s). To read the attribute, you must cast the `Sequence` object[3].

[1] Refer to ExecutionError Class Description on page 78.
[2] Refer to DataStreams Collection Description on page 77.
[3] Refer to Getting the Run Time of an RTT Sequence (Real-Time Testing Guide 🕮).

Methods

The following methods are part of the class:

Method	Purpose
Continue	To continue the sequence execution at the same point of time where it was paused. Refer to Continue Method on page 70.
Pause	To pause a running RTT sequence. Refer to Pause Method on page 71.
Remove	To remove an RTT sequence from the simulation platform. Refer to Remove Method on page 71.
Run	To start an RTT sequence on the simulation platform. Refer to Run Method on page 72.
Stop	To stop a running RTT sequence. Refer to Stop Method on page 73.

Related topics

References

Common Constants of rttmanagerlib.. 84

Continue Method

Class

Sequence

Syntax

```
OBJ.Continue()
```

Purpose

To continue the sequence execution at the point where it was paused. The new RTT sequence state after `Continue` is `rttmanager.constants.sesRunning`.

Parameter

–

Return value	–
Related topics	Basics States of RTT Sequences (Real-Time Testing Guide 🕮)

Pause Method

Class	Sequence
Syntax	`OBJ.Pause()`
Purpose	To pause a running RTT sequence. The new RTT sequence state after `Pause` is `rttmanager.constants.sesPaused`.
Parameter	–
Return value	–
Related topics	Basics States of RTT Sequences (Real-Time Testing Guide 🕮)

Remove Method

Class	Sequence
Syntax	`OBJ.Remove()`
Purpose	To remove an RTT sequence from the simulation platform.

Parameter

–

Return value

–

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

Run Method

Class

Sequence

Syntax

```
OBJ.Run(RunParameter)
```

Purpose

To start an RTT sequence on the simulation platform. The new RTT sequence state after `Run` is `rttmanager.constants.sesRunning`.

Description

An RTT sequence can be started if it has one of the following states:

- New
- Paused
- Stopped
- Terminated

When an RTT sequence is started with the Paused, Stopped, or Terminated state, its namespace is maintained. The RTT sequence is not initialized but starts directly with executing the `MainGenerator` function.

The RTT sequences are executed according to their priority in the sequence list.

Run with parameter You can pass one parameter to the RTT sequence when starting the sequence execution. Empty strings in the parameter list are deleted and not passed. To pass the parameter list, the `MainGenerator()` function must contain an `(*args)` parameter.

If you pass parameters without the `(*args)` parameter, an exception is raised in the RTT sequence.

You can pass only one parameter. If you want to pass multiple objects, use a list or a tuple.

Parameter

The method uses the following parameter:

Parameter	Type	Description
RunParameter	Python object	Optional parameter. Python object of the parameter list passed to the RTT sequence when starting the sequence execution.

Return value

–

Example

For an example of running an RTT sequence with parameters, refer to Starting RTT Sequences with Arguments in Python Scripts (Real-Time Testing Guide 📖).

Related topics

Basics

Starting RTT Sequences with Arguments in Python Scripts (Real-Time Testing Guide 📖)
States of RTT Sequences (Real-Time Testing Guide 📖)

References

RunAll Method.. 61

Stop Method

Class

Sequence

Syntax

```
OBJ.Stop()
```

Purpose

To stop a running RTT sequence. The new RTT Sequence state after `Stop` is `rttmanager.constants.sesStopped`.

Parameter

–

Return value

–

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 🕮)

SequenceEvents

Where to go from here

Information in this section

SequenceEvents Class Description..74
To handle events of an RTT sequence.

OnHostCall Method...76
To receive a host call from the real-time platform and return a result.

SequenceEvents Class Description

Syntax

See Example below.

Purpose

To handle events of an RTT sequence.

Description

An RTT sequence can trigger events which you can evaluate in the host PC script. You can also implement event handling in the script. For an example, refer to Handling OnHostCall Events of an RTT Sequence in Python Scripts (Real-Time Testing Guide 🕮).

Attributes

–

Methods

The following methods for the events are defined in the class:

Method	Description
OnHostCall	The host script got a Python data object from an RTT sequence. The return value of OnHostCall is sent to the RTT sequence. Refer to OnHostCall Method on page 76.

If the EventObject is destroyed, no events are output.

Tip

Use rttutilities.RTTSleep() to wait for events, for example, OnWrite events after executing Sequence.Run(). If you use other Sleep functions, for example, win32api.Sleep(), the events are not output and it may lead to a deadlock.

Example

```
import rttmanagerlib
class RTTMHostCallEvents(rttmanagerlib._IRTSequenceEvents):
    def __init__(self, Sequence, BoardName):
        # Call base class constructor to connect to event source
        rttmanagerlib._IRTSequenceEvents.__init__(self, Sequence)
        self.CurrentBoardName = BoardName
    def OnHostCall(self, *Data):
        """Method OnHostCall"""
        ReturnResultToRT = []
        for Element in Data:
            if isinstance(Element, str):
               print("OnHostCall:" + str(Element))
               ReturnResultToRT.append("String '%s' received on host."\
                    %Element)
        for Element in Data:
            if isinstance(Element, str):
               print("OnHostCall:" + str(Element))
               ReturnResultToRT.append("String '%s' received on host."\
                    %Element)
            else:
               print("%s '%s' of type '%s'." %("OnHostCall:",\
                    Element, type(Element)))
        return ReturnResultToRT
def main():
    SequenceEvents = None
    rttm = rttmanagerlib.RealTimeTestManagerServer()
    try:
        Board = rttm.AccessBoard("127.0.0.1/MyApp")
        # Connect to sequences event handle
        SequenceEvents = RTTMSequenceEvents(Board.Sequences)
        #...
        rttutilities.RTSleep(10) # wait for 10 sec for host call event
    finally:
        if SequenceEvents:
            # Disconnect from sequences event handle
            SequenceEvents.close()
            SequenceEvents = None
        Board = None
        rttm = None
#------------------------------------------------------------
# Module main block
#------------------------------------------------------------
if __name__ == '__main__':
    main()
```

Related topics

Basics

Handling OnHostCall Events of an RTT Sequence in Python Scripts (Real-Time Testing Guide 📖)

OnHostCall Method

Class

SequenceEvents

Syntax

```
OnHostCall(self, args)
```

Purpose

To receive a Python data object from an RTT sequence and send the return value to the RTT sequence.

Parameter

The method uses the following parameter:

Parameter	Type	Description
args	Tuple	Tuple with Python objects. The tuple is filled in an RTT sequence.

Return value

The method returns a value of the following type:

ReturnValue	Type	Description
ReturnResultToRT	Tuple	The OnHostCall method can return one or more objects of arbitrary type. These objects are returned to the hostcall caller in a tuple. This tuple is passed to hostcall.Hostcall(...) as the first parameter.

Note

The send and return values must be restorable with a Python cPickle module. For more details, refer to the official *Python* documentation.

Related topics

Basics

Handling OnHostCall Events of an RTT Sequence in Python Scripts (Real-Time Testing Guide 📖)

DataStreams

DataStreams Collection Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Sequence = Board.Sequences[0]
DataStreamFileName = Sequence.DataStreams[0].FileName
```

Purpose

To get information on the datastreams which are used in an RTT sequence.

Attribute

The following attributes are part of the class.

Attribute	Type	Purpose
FileName	String	To get the file name of the MAT file used for data streaming.
Name	String	To get the name of the datastream. The name is specified automatically: *DataStream_<n>* where <n> is 0, 1, 2 ...
SequenceName	String	To get the name of the RTT sequence which uses the datastream.

Methods

–

Related topics

Basics

Basics of Data Replay Using ASAM MDF (MF4) Files (Real-Time Testing Guide 🕮)
Basics of Data Replay Using MAT Files (Real-Time Testing Guide 🕮)

ExecutionError

ExecutionError Class Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
ExecutionError = Board.Sequences[0].LastExecutionError
print("Stack: ", ExecutionError.Stack)
print("Type: ", ExecutionError.Type)
print("Value: ", ExecutionError.Value)
```

Purpose

To get information on errors occurring during sequence execution.

If no error occurred, `LastExecutionError` returns `None`.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
Stack	String	To get traceback information on the error
Type	String	To get the exception type, for example, ZeroDivisionError
Value	String	To get exception information, for example, integer division or modulo by zero

Methods

–

Related topics

Basics

Managing RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

Variables (Collection)

Where to go from here

Information in this section

Variables Class (Collection) Description...79
To manage the collection of dynamic variables on a simulation platform.

Item Method..80
To return a dynamic variable by index.

Variables Class (Collection) Description

Syntax

```
import rttmanagerlib
rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Board.Variables
```

Purpose

To manage the collection of dynamic variables on a simulation platform.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Count	Integer	To get the number of items in the collection

Methods

The following methods are part of the class:

Method	Purpose
Item	To return a dynamic variable by index or name.

Related topics

References

DynamicVariable Class...160

Item Method

Class

Variables

Syntax

```
RetVal = OBJ.Item(Index)
```

Purpose

To return a dynamic variable by index.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Index	Integer or String	Index or Name of the dynamic variable

Return value

The method returns a value of the following type:

Type	Description
Variable	To manage a dynamic variable.

Related topics

References

Variable Class Description..83

VariablesEvents

Where to go from here

Information in this section

VariablesEvents Class Description...81
To handle the events of the dynamic variables.

OnAdd Method..82
To output the added dynamic variable object.

VariablesEvents Class Description

Syntax

See Example below.

Purpose

To handle the events of the dynamic variables.

Description

Dynamic variables can trigger events which you can evaluate in the host PC script. You can also implement event handling in the script. Refer to Basics on Dynamic Variables (Real-Time Testing Guide 📖).

If the `EventObject` is destroyed, no events are output.

Tip

Use `rttutilities.RTTSleep()` to wait for events, for example, `OnWrite` events after executing `Sequence.Run()`. If you use other Sleep functions, for example, `win32api.Sleep()`, the events are not output and it may lead to a deadlock.

Attributes

–

Methods

The following methods for the events are defined in the class:

Method	Description
OnAdd	A dynamic variable was created on the simulation platform.

Example

```
import rttmanagerlib
class RTTMVariablesEvents(rttmanagerlib._IRTVariablesEvents):
    def __init__(self, EventSource = None, Parent = None):
        # Call base class constructor to connect to event source
        rttmanagerlib._IRTVariablesEvents.__init__(self, EventSource)
        self.Parent = Parent
    def OnAdd(self, Variable):
        """Method OnAdd"""
        Variable = rttmanagerlib.IRTVariable(Variable)
        print(VARIABLE_PREFIX)
        print(VARIABLE_PREFIX + r"OnAdd: New RTT variable '%s' created." \
            %(Variable.Name))
        print(VARIABLE_PREFIX + r"  Name:          '%s'" \
            %(Variable.Name))
        print(VARIABLE_PREFIX + r"  Sequence name: '%s'" \
            %(Variable.SequenceName))
```

```
def main():
    VariablesEvents = None
    rttm = rttmanagerlib.RealTimeTestManagerServer()
    try:
        Board = rttm.AccessBoard("127.0.0.1/MyApp")
        # Connect to variables event handle
        VariablesEvents = RTTMVariablesEvents(Board.Variables)
        (...)
    finally:
        if VariablesEvents:
            # Disconnect from variables event handle
            VariablesEvents.close()
            VariablesEvents = None
            Board = None
            rttm = None
#------------------------------------------------------------
# Module main block
#------------------------------------------------------------
if __name__ == '__main__':
    main()
```

Related topics

References

OnAdd Method.. 82

OnAdd Method

Class

SequencesEvents

Syntax

```
OnAdd(Variable)
```

Purpose

To output the added dynamic variable object.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Variable	Variable[1]	The added dynamic variable object

[1] Refer to Variable Class Description on page 83.

Return value

–

Related topics

References

Variable Class Description.. 83

Variable

Variable Class Description

Syntax

```
import rttmanagerlib

rttm = rttmanagerlib.RealTimeTestManagerServer()
Board = rttm.AccessBoard("127.0.0.1/MyApp")
Variable = Board.Variables[0]
```

Purpose

To manage a dynamic variable.

Description

Dynamic variables can be configured during run-time.

You can remove dynamic variables only by reloading the real-time application.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Name	String	To get the name of the dynamic variable
SequenceName	String	To get the name of the RTT sequence in which you created the dynamic variable
Value	Float	To get/set the value of the dynamic variable from/to the host PC

Methods

–

Related topics

References

DynamicVariable Class..160

Common Constants

Common Constants of rttmanagerlib

List of constants

You can use constants to specify the execution order of RTT sequences and to change from one state to the other. The following constants are used to specify the common attributes of rttmanagerlib:

Constants for specifying the script channel

Value	Description
scPreComputation	In a sampling step, the RTT sequence is executed before the model simulation is executed by the real-time application.
scPostComputation	In a sampling step, the RTT sequence is executed after the model simulation is executed by the real-time application.

For an illustration of Pre- and PostComputation, refer to *Channel* in Basics on Executing RTT Sequences (Real-Time Testing Guide 📖).

Constants for RTT sequence states

Value	Description
sesError	Error when creating or executing RTT sequences
sesNew	New RTT sequence was created.
sesPaused	RTT sequence is paused.
sesRunning	RTT sequence is running.
sesStopped	RTT sequence was stopped.
sesTerminated	RTT sequence was executed completely without errors.

Related topics

Basics

States of RTT Sequences (Real-Time Testing Guide 📖)

rttutilities Module

Introduction

This module provides functions for on the host PC that are useful for real-time testing but not available in the standard Python libraries.

Where to go from here

Information in this section

GetTraceBackString Function......86
To return a string representation of the last Python exception message.

RTTSleep Function Description......87
To suspend code execution for a specified number of seconds.

GetTraceBackString Function

Syntax

```
import rttutilities

rttutilities.GetTraceBackString()
```

Purpose

To return a string representation of the last Python exception message.

Description

The string representation consists of Type, Value, and Traceback. Refer to the `traceback` standard Python module.

Parameter

–

Return value

The function returns a value of the following type:

Type	Description
String	The Python traceback information

Related topics

Basics

Implementing an Exception Handling (Real-Time Testing Guide 📖)

RTTSleep Function Description

Syntax

```
import rttutilities
rttutilities.RTTSleep(Seconds)
```

Purpose

To suspend code execution for a specified number of seconds.

Description

The function is a non-blocking sleep function which suspends code execution for a specified number of seconds. In contrast to `time.sleep()`, `RTTSleep()` allows handling events, for example, OnError, during sleep.

The argument can be a floating-point number to indicate a more precise sleep time. The actual suspension time can be less than requested because any caught signal will terminate the `RTTSleep()` following execution of the signal's catching routine. The suspension time can also be longer than requested by an arbitrary amount because of the scheduling of other activities in the system.

Parameter

The function uses the following parameter:

Parameter	Type	Description
Seconds	Float	The number of seconds to suspend execution.

Return value

–

Related topics

Basics

Using Sleep() Function (Real-Time Testing Guide 📖)

dSPACE Python Modules for Implementing RTT Sequences

Introduction

The Python interpreter on the simulation platform contains the rttlib package. You can import the contained modules in the RTT sequence.

Where to go from here

Information in this section

rttlib.canlib Module....91
To provide functions for sending and receiving CAN messages with the RTT sequences.

rttlib.dscanapilib Module....111
To provide functions for sending and receiving CAN messages with the RTT sequences.

rttlib.datastream Module....141
Provides a class for streaming data from MAT files and MDF4 files on the host PC to an RTT sequence to stimulate variables.

rttlib.dynamicvariable Module....160
Provides a class to represent an object for accessing dynamic variables from an RTT sequence and from the host PC. Dynamic variables can be created during the real-time application's run time.

rttlib.errors Module....163
For exception handling on the real-time platform.

rttlib.dsethernetapilib Module....164
To provide functions for sending and receiving Ethernet raw frames with the RTT sequences on the real-time platform.

rttlib.globalvariables Module....187
To use global variables in different RTT sequences.

rttlib.hostcall Module..188
To provide functions for sending data to a registered Python script on the host PC.

rttlib.rs232lib Module..190
To provide functions for sending and receiving data via an RS232 interface of the real-time platform.

rttlib.scheduler Module..198
To execute concurrent operations in an RTT sequence.

rttlib.utilities Module..202
To provide functions for common real-time operations.

rttlib.variable Module..212
Provides a class to represent a variable object for accessing Simulink variables from an RTT sequence.

rttlib.watcherlib Module..216
Provides a class to check conditions according to the ASAM GES standard.

rttlib.canlib Module

Introduction

This module provides functions for sending and receiving CAN messages with the RTT sequences.

Where to go from here

Information in this section

rttlib.canlib Module Quick Reference....92

controllers....93
To manage the collection of controllers on the real-time platform.

canmmbaselib....94
To specify the message format.

canmmlib....95
To provide functions for sending and receiving CAN messages with the RTT sequences.

controller....98
To access the controller of a channel.

channel....100
To access the specified channel.

message....102
To access the specified message.

messagerx....107
To get information on an RX message.

messagetx....108
To set data of a TX message.

canmmerror....109
To generate an exception if a timeout occurs when sending or receiving messages.

Information in other sections

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)
You can use the rttlib.canlib module when the Simulink model uses the RTI CAN MultiMessage Blockset.

rttlib.canlib Module Quick Reference

Introduction

Object information of the canlib module is summarized in a compact table, which provides a quick overview of the available objects, object dependencies, attributes and methods.

Overview of the canlib Object Model

Introduction

The object model overview of the canlib module gives a quick overview of object dependencies, and available object attributes and methods.

Symbols

The following symbols are used in the object model overview:

Symbol	Description
●	Method, function
▣	Attribute (property, class)
▦	Collection
0, 1, 2, …	Level of dependency (0, 1, 2, …)
⊗	Read only

canlib

The following table gives an overview of the canlib's object model:

canlib	0
▦ controllers controllers	1
▣ canmmbaselib canmmbaselib	2
▣ Constant ftSTD	3
▣ Constant ftEXT	
▣ Constant ftFDSTD	
▣ Constant ftFDEXT	
▣ Constant ftFDSTDALTBR	
▣ Constant ftFDEXTALTBR	
▣ canmmlib canmmlib	2
▣ controller controller	3
▣ string Name	
▣ channel channel	4
▣ message message	5

canlib	0
int DLC	6
int Format	
int ID	
messagerx RX	
longlong Data	7
float TimeStamp	
float DeltaTime	
int Counter	
int Status	
messagetx TX	6
int Counter	7
longlong Data	
float DelayTime	
int IsReady	
None Receive()	6
Generator object yield ReceiveGen(TimeOutSteps = 10)	
None Transmit()	
Generator object yield TransmitGen(TimeOutSteps = 10)	
message GetRawMessage()	5
channel GetChannel(ChannelIndex)	4
canmmerror canmmerror[1]	3
controller GetController(ControllerTRCPathName)	

[1] See RTTException on page 163

Related topics

Basics

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)

controllers

Purpose

To manage the collection of controllers on the real-time platform.

controllers Class Description

Syntax

```
from rttlib.canlib import controllers
```

Purpose

To manage the collection of controllers on the real-time platform.

Attributes

–

Methods

–

Related topics

Basics

Accessing the CAN Bus with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

canmmbaselib

Purpose

To specify the message format.

canmmbaselib Class Description

Syntax

```
Message1 = Channel.GetRawMessage()
Message1.Format = canmmlib.canmmbaselib.ftSTD
```

Purpose

To specify the message format.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
ftSTD	Constant	To set the format for CAN message, IDs with a maximum of 11 bit (CAN 2.0 A, standard frame format).

Attribute	Type	Purpose
ftEXT	Constant	To set the format for CAN message, IDs with a maximum of 29 bit (CAN 2.0 B, extended frame format).
ftFDSTD	Constant	To set the format for CAN FD message, IDs with a maximum of 11 bit.
ftFDEXT	Constant	To set the format for CAN FD message, IDs with a maximum of 29 bit.
ftFDSTDALTBR	Constant	To set the format for CAN FD message using a higher bit rate, IDs with a maximum of 11 bit.
ftFDEXTALTBR	Constant	To set the format for CAN FD message using a higher bit rate, IDs with a maximum of 29 bit.

Methods

–

Related topics

Basics

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)

References

Overview of the canlib Object Model..92

canmmlib

Purpose

To provide functions for sending and receiving CAN messages with the RTT sequences.

Where to go from here

Information in this section

canmmlib Class Description..96
To describe the class and its attributes.

GetController Method..96
To get the controller of the CAN bus.

canmmlib Class Description

Syntax

```
from rttlib.canlib.controllers import canmmlib
```

Purpose

To provide functions for sending and receiving CAN messages with the RTT sequences on the real-time platform.

Attributes

–

Methods

The following method is part of the class:

Method	Purpose
GetController	To get the controller for the current TRC path. Refer to GetController Method on page 96.

Related topics

Basics

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)

GetController Method

Class

canmmlib

Syntax

```
Controller = canmmlib.GetController(ControllerTRCPathName)
```

Purpose

To get the controller of the CAN bus.

Parameter

The method uses the following parameter:

Parameter	Type	Description
ControllerTRCPathName	String	The TRC path of the controller. This path is displayed in ControlDesk. For a CAN bus, the TRC path consists of `BusSystem/CAN` and the CAN bus name which is specified on the

Parameter	Type	Description
		General Settings page of the RTICANMM MainBlock. In multiprocessor systems, you must use the path of the CPU where the RTT sequence is running.

Return value

The method returns a value of the following type:

Type	Description
controller[1]	The controller for the selected CAN bus specified by the TRC path.

[1] Refer to controller on page 98.

Example

The example shows how to access the controller of the CAN bus named `Chassis`. The controller has only one channel.

```python
from rttlib.canlib.controllers import canmmlib

MyController = None
MyCANBus = r"BusSystems/CAN/Chassis"
MyController = canmmlib.GetController(MyCANBus)
MyChannel = MyController.GetChannel()
```

Related topics

Basics

Accessing the CAN Bus with the rttlib.canlib Module (Real-Time Testing Guide 📖)

References

General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 📖)

controller

Purpose

To access the controller of a channel.

Where to go from here

Information in this section

controller Class Description.. 98
To describe the class and its attributes.

GetChannel Method.. 99
To get the channel for the controller by index.

controller Class Description

Syntax

```
MyController = None
MyCANBus = r"BusSystems/CAN/Chassis"
MyController = canmmlib.GetController(MyCANBus)
```

Purpose

To access the controller of a channel.

Limitation

You can access only the CAN or CAN FD controllers which are on the CPU where the RTT sequence is running. Accessing a CAN or CAN FD controller on a remote CPU in a multiprocessor system is not supported.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
Name	String	To get the name of the controller
IsCANFDSupportEnabled	Integer	To check whether CAN FD support is enabled for the CAN controller in this real-time application. ▪ 0: CAN FD support is *not* enabled. ▪ 1: CAN FD support is enabled.

Methods

The following method is part of the class:

Method	Purpose
GetChannel	To get the channel for the current controller. Refer to GetChannel Method on page 99.

Related topics

Basics

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)

GetChannel Method

Class

controller

Syntax

```
Channel = controller.GetChannel(ChannelIndex)
```

Purpose

To get the channel for the controller by index.

Parameter

The method uses the following parameter:

Parameter	Type	Description
ChannelIndex	String	The index of the TRC path of the channel. At the moment, each supported controller has one channel. This means the value is 0.

Return value

The method returns a value of the following type:

Type	Description
channel[1]	To access the specified channel.

[1] Refer to channel Class Description on page 100.

Example

The example shows how to access the controller of the CAN bus named `Chassis`. The controller has only one channel.

```
from rttlib.canlib.controllers import canmmlib

MyController = None
MyCANBus = r"BusSystems/CAN/Chassis"
MyController = canmmlib.GetController(MyCANBus)
MyChannel = MyController.GetChannel()
```

Related topics

Basics

Accessing the CAN Bus with the rttlib.canlib Module (Real-Time Testing Guide 📖)

channel

Purpose

To access the specified channel.

Where to go from here

Information in this section

channel Class Description..100
To describe the class and its attributes.

GetRawMessage Method..101
To reserve an experimental message for the current channel.

channel Class Description

Syntax

```
MyController = None
MyCANBus = r"BusSystems/CAN/Chassis"
MyController = canmmlib.GetController(MyCANBus)
MyChannel = MyController.GetChannel()
```

Purpose

To access the specified channel.

Attributes

–

Methods

The following methods are part of the class:

Method	Purpose
GetRawMessage	To get the raw message for the current channel. Refer to GetRawMessage Method on page 101.

Related topics

Basics

Handling CAN Messages Using the rttlib.canlib Module (Real-Time Testing Guide 📖)

GetRawMessage Method

Class

channel

Syntax

```
OBJ.GetRawMessage()
```

Purpose

To reserve an experimental message for the current channel.

Description

This method reserves a free experimental message for the channel. The maximum number of experimental messages is specified on the **Experimental Software** page of the **RTICANMM MainBlock**.

As the number of experimental messages is limited, you should always delete the message object when it is no longer used. If an exception in an RTT sequence occurs before the message object is deleted, the experimental message is still reserved. In this case, you must delete the RTT sequence on the platform.

Parameter

–

Return value

The method returns a value of the following type:

Type	Description
message[1]	The raw message for the current channel.

[1] Refer to message Class Description on page 102.

Related topics

Basics

Receiving CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)
Sending CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

References

Experimental Software Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
RTICANMM MainBlock (RTI CAN MultiMessage Blockset Reference 🕮)

message

Purpose

To access the specified message.

Where to go from here

Information in this section

message Class Description 102
To describe the class and its attributes.

Receive Method 104
To receive the message.

ReceiveGen Method 105
To wait for the message reception for a specified number of sampling steps.

Transmit Method 106
To transmit the message (only for advanced user).

TransmitGen Method 106
To wait for the message transmission for a specified number of sampling steps.

message Class Description

Syntax

```
Message = Channel.GetRawMessage()
```

Purpose

To access the specified message.

Description

The message allocates the experimental message (channel resource). It must be cleared if the resource is not used because the maximum number of experimental messages is limited.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
DLC	Integer	To get the number of bytes of the current message (DLC: data length code). For CAN messages, the maximum data length is 8 bytes. For CAN FD messages, the maximum data length is 64 byte. Only the following DLC values are valid: 0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 16, 20, 24, 32, 48, 64
Format	Integer	To get the message identifier format. The format can be 0: `canmmlib.canmmbaselib.ftSTD` (Standard frame format) `1: canmmlib.canmmbaselib.ftEXT` (Extended frame format) `2: canmmlib.canmmbaselib.ftFDSTD` (Standard CAN FD frame format) `3: canmmlib.canmmbaselib.ftFDEXT` (Extended CAN FD frame format) `6: canmmlib.canmmbaselib.ftFDSTDALTBR` (Standard CAN FD frame format using a higher bit rate) `7: canmmlib.canmmbaselib.ftFDEXTALTBR` (Extended CAN FD frame format using a higher bit rate)
ID	Integer	To get the CAN identifier. The length is 11 bit for a CAN message in standard frame format and 29 bit for a CAN message in extended frame format.
RX	messagerx[1)]	To get the receive path of the message (read-only). When the message is received, the data is saved here.
TX	messagetx[2)]	To get the transmit path of the message (read-only). The data to be transmitted can be stored here.

[1)] Refer to messagerx Class Description on page 107.

[2)] Refer to messagetx Class Description on page 108.

Methods

The following methods are part of the class:

Method	Purpose
Receive	To receive the message. Refer to Receive Method on page 104.
ReceiveGen	To receive a message with a specified timeout. Refer to ReceiveGen Method on page 105.
Transmit	To transmit the message. Refer to Transmit Method on page 106.
TransmitGen	To transmit a message and wait for the message transmission. Refer to TransmitGen Method on page 106.

Related topics

Basics

Receiving CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)
Sending CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

References

GetRawMessage Method.. 101

Receive Method

Class

message

Syntax

```
OBJ.Receive()
```

Purpose

To receive the CAN message.

Description

The `messagerx.data` attribute is automatically filled with raw data when the CAN message with the specified ID is received. The content of the current sampling step is given back by the `Receive` method, which is non-blocking. It is therefore not necessary to call the `Receive()` method because this version of Real-Time Testing handles only raw data. These methods will become necessary in a future version if received data is encoded on the basis of a DBC file, for example.

Note that the transmission of a CAN message also automatically fills the receive buffer with the transmitted raw data (loopback mechanism).

Parameter

–

Return value

–

Related topics

Basics

Receiving CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

ReceiveGen Method

Class

message

Syntax

```
yield OBJ.ReceiveGen(TimeOutSteps = 10)
```

Purpose

To wait for the message reception for a specified number of sampling steps.

Description

The `ReceiveGen` method blocks until the corresponding message is received (with sample steps as timeout). If the message is not received within the specified number of sampling steps, `canmmerror` is raised. In this case, the message object contains the latest received message data (received before the execution of `ReceiveGen`).

Note that the transmission of a CAN message also automatically fills the receive buffer with the transmitted raw data (loopback mechanism).

Parameter

The method uses the following parameter:

Parameter	Type	Description
TimeOutSteps	Integer	The number of sample steps after which canmmerror is raised if the RTT sequence does not receive the message. The default value is 10 sample steps.

Return value

–

Related topics

Basics

Receiving CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

References

canmmerror Class Description..110

Transmit Method

Class

message

Syntax

```
OBJ.Transmit()
```

Purpose

To transmit the message (only for advanced user).

Description

The method transmit the CAN message. For the transmission, it requires two sampling steps.

```
Obj.Transmit()
yield None
yield None
Obj.Transmit()
```

Parameter

–

Return value

–

Related topics

Basics

Sending CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

TransmitGen Method

Class

message

Syntax

```
yield TransmitGen(TimeOutSteps = 10)
```

Purpose

To wait for the message transmission for a specified number of sampling steps.

Description

If the message cannot be transmitted within the specified number of sampling steps, `canmmerror` is raised.

Parameter

The method uses the following parameter:

Parameter	Type	Description
TimeOutSteps	Integer	The number of sampling steps after which canmmerror is raised if the RTT sequence cannot transmit the message. The default value is 10 sampling steps.

Return value

–

Related topics

Basics

Sending CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

References

canmmerror Class Description..110

messagerx

Purpose

To get information on an RX message.

messagerx Class Description

Syntax

```
RXCounter = Msg.RX.Counter
Msg.Receive()
```

Purpose

To get information on an RX message.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
Data	LongLong	To get the data which you receive (read-only): ▪ For CAN messages this is a value with a maximum length of 64 bit.

Attribute	Type	Purpose
		▪ For CAN FD messages this is a value with a maximum length of 64 bytes.
TimeStamp	Float	To get the point in time in seconds at which the message was received (read-only)
DeltaTime	Float	To get the difference in seconds between the points in time at which the current and the previous message were received (read-only)
Counter	Integer	To get the number of received messages (read-only)
Status	Integer	To get the status of an RX message (read-only). ▪ 0: The message is not received. ▪ 1: The message was received in the current sample step. The value is set to 0 in the next sampling step.

Methods

–

Related topics

Basics

Receiving CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 🕮)

messagetx

Purpose

To set data of a TX message.

messagetx Class Description

Syntax

```
Counter = Msg.TX.Counter
Msg.Transmit()
```

Purpose

To set data of a TX message.

Attributes

The following attributes are part of the class.

Attribute	Type	Purpose
Counter	Integer	To get the number of transmitted messages 📖
Data	LongLong	To get/set data to transmit: ▪ For CAN messages this is a value with a maximum length of 64 bit. ▪ For CAN FD messages this is a value with a maximum length of 64 bytes.
DelayTime	Float	To get/set the delay time of the transmission. Transmission of the TX message starts after the delay time elapsed.
IsReady	Integer	To get the transmission status of the TX message (read-only). ▪ 1: The message data is sent to the CAN controller. The message data can be configured for the next transmission. ▪ 0: The message data was not sent yet.

Methods

–

Example

The example shows how the attributes of a TX message in standard frame format are set.

```
MyTXMessage = MyChannel.GetRawMessage()
MyTXMessage.Format = canmmlib.canmmbaselib.ftSTD
MyTXMessage.ID = 0x123
MyTXMessage.DLC = 4
MyTXMessage.TX.Data = 0x10203040
MyTXMessage.Transmit()
```

Related topics

Basics

Sending CAN Messages with the rttlib.canlib Module (Real-Time Testing Guide 📖)

canmmerror

Purpose

To generate an exception if a timeout occurs when sending or receiving messages.

canmmerror Class Description

Syntax

```
Counter = Msg.RX.Counter
# wait for message received
CurrentSteps = 0
while Counter == Msg.RX.Counter:
    yield None
    if  CurrentSteps == TimeOutSteps:
        raise canmmerror("Message was not received. ID: " \
                + hex(Msg.ID) + ", timeoutsteps: " + str(TimeOutSteps))
    CurrentSteps = CurrentSteps + 1
```

Purpose

To generate an exception if a timeout occurs when sending or receiving messages.

Attributes

–

Methods

–

Related topics

References

ReceiveGen Method.......... 105
TransmitGen Method.......... 106

rttlib.dscanapilib Module

Introduction

To provide functions for sending and receiving CAN messages with the RTT sequences.

Where to go from here

Information in this section

dscanapilib..111
To send and receive CAN messages via RTT sequences.

BitTimingParameters..134
To get/set all the necessary parameters for the baud rate.

BusInfo..135
To get and summarize information about the bus.

BusStatistics..136
To hold all bus statistics information of a channel for a specified time.

CanMessage..137
To create CAN messages to be sent or to read received messages.

ChannelInfo...139
To get information about the available CAN channels.

Information in other sections

Handling CAN Messages Using the rttlib.dscanapilib Module (Real-Time Testing Guide 📖)
You can use the rttlib.dscanapilib module when the Simulink model does not use the RTI CAN MultiMessage Blockset.

dscanapilib

Purpose

To send and receive CAN messages via RTT sequences.

Where to go from here

Information in this section

Class Description (dscanapilib)...113
To describe the class and its attributes.

ActivateChannel........115
To activate a CAN channel.

DeactivateChannel........116
To deactivate a CAN channel.

EnableBusStatistics........116
To enable or disable the bus statistic frames in the receive queue.

EncodeBusStatistics........117
To encode CAN bus statistics values of the CAN messages in the receive queue.

FlushReceiveQueue........118
To flush the receive queue of the specified CAN channel.

FlushTransmitQueue........119
To flush the transmit queue of the specified CAN channel.

GetAvailableChannels........120
To get information about the available CAN channels.

GetBaudrate........120
To get the baud rate for a CAN channel.

GetBusInfo........121
To get information of the bus of a CAN channel.

GetBusType........122
To get the bus type for a CAN channel.

GetErrorText........123
To get the error text corresponding to a specified error code.

GetHardwareTime........123
To get the hardware time of a specific channel.

GetHardwareTimeResolution........124
To get the hardware time resolution of a specific channel.

InitChannel........125
To initialize a specific channel and put it in a usable state.

ReadReceiveQueue........126
To read the CAN messages from the receive queue of a CAN channel.

RegisterChannel........127
To register a CAN channel and get a handle for subsequent function calls.

ResetHardwareTime........128
To reset the hardware time of a CAN interface.

SetAcceptance........129
To specify the acceptance for a CAN channel to filter incoming CAN messages by their identifiers.

SetBaudrate..130
To set the baud rate for a channel.

SetChannelOutput..131
To specify whether a CAN channel operates in normal or silent mode.

SetTransmitAcknowledge...132
To activate or deactivate the transmit acknowledge for a CAN channel.

TransmitMessages..132
To copy CAN messages to the send buffer for transmission.

UnregisterChannel..133
To unregister a CAN channel that is currently in use.

Class Description (dscanapilib)

Syntax

```
from rttlib import dscanapilib
```

Purpose

To send and receive CAN messages via RTT sequences.

Description

The `rttlib.dscanapilib` module is an alternative to `rttlib.canlib`.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
BitTimingParameters	BitTimingParameters[1]	To get/set all the necessary parameters for the baud rate.
BusInfo	BusInfo[2]	To get and summarize information about the bus.
CanMessage	CanMessage[3]	To create CAN messages to be sent or to read received messages.
ChannelInfo	ChannelInfo[4]	To get information about the available CAN channels.

[1] Refer to BitTimingParameters on page 134.
[2] Refer to BusInfo on page 135.
[3] Refer to CanMessage on page 137.
[4] Refer to ChannelInfo on page 139.

Methods

The class contains the following methods:

Method	Purpose
ActivateChannel	To activate a CAN channel. Refer to ActivateChannel on page 115.
DeactivateChannel	To deactivate a CAN channel. Refer to DeactivateChannel on page 116.
FlushReceiveQueue	To flush the receive queue of the specified CAN channel. Refer to FlushReceiveQueue on page 118.
FlushTransmitQueue	To flush the transmit queue of the specified CAN channel. Refer to FlushTransmitQueue on page 119.
GetAvailableChannels	To get information about the available CAN channels. Refer to GetAvailableChannels on page 120.
GetBaudrate	To get the baud rate for a CAN channel. Refer to GetBaudrate on page 120.
GetBusInfo	To get information of the bus of a CAN channel. Refer to GetBusInfo on page 121.
GetBusType	To get the bus type of a CAN channel. Refer to GetBusType on page 122.
GetErrorText	To get the error text corresponding to a specified error code. Refer to GetErrorText on page 123.
GetHardwareTime	To get the hardware time of a specific channel. Refer to GetHardwareTime on page 123.
GetHardwareTimeResolution	To get the hardware time resolution of a specific channel. Refer to GetHardwareTimeResolution on page 124.
InitChannel	To initialize a specific channel and put it in a usable state. Refer to InitChannel on page 125.
ReadReceiveQueue	To read the CAN messages from the receive queue of a CAN channel. Refer to ReadReceiveQueue on page 126.
RegisterChannel	To register a CAN channel and get a handle for subsequent function calls. Refer to RegisterChannel on page 127.
ResetHardwareTime	To reset the hardware time of a CAN interface. Refer to ResetHardwareTime on page 128.
SetAcceptance	To specify the acceptance for a CAN channel to filter incoming CAN messages by their identifiers. Refer to SetAcceptance on page 129.
SetBaudrate	To set the baud rate for a channel. Refer to SetBaudrate on page 130.
SetChannelOutput	To specify whether a CAN channel operates in normal or silent mode. Refer to SetChannelOutput on page 131.
SetTransmitAcknowledge	To activate or deactivate the transmit acknowledge for a CAN channel. Refer to SetTransmitAcknowledge on page 132.
TransmitMessages	To copy CAN messages to the send buffer for transmission. Refer to TransmitMessages on page 132.
UnregisterChannel	To unregister a CAN channel that is currently in use. Refer to UnregisterChannel on page 133.

Related topics

Basics

Handling CAN Messages Using the rttlib.dscanapilib Module (Real-Time Testing Guide 🕮)

ActivateChannel

Class

dscanapilib

Syntax

```
dscanapilib.ActivateChannel(ChannelHandle)
```

Purpose

To activate a CAN channel.

Description

The CAN channel must be registered and initialized before it can be activated, refer to RegisterChannel on page 127 and InitChannel on page 125.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be activated.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during activation, None is returned.

Related topics

References

Class Description (dscanapilib)........ 113
InitChannel........ 125
RegisterChannel........ 127

DeactivateChannel

Class

dscanapilib

Syntax

```
dscanapilib.DeactivateChannel(ChannelHandle)
```

Purpose

To deactivate a CAN channel.

Description

The CAN channel must be activated before it can be deactivated, refer to ActivateChannel on page 115.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be deactivated.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during deactivation, `None` is returned.

Related topics

References

Class Description (dscanapilib)...113

EnableBusStatistics

Class

dscanapilib

Syntax

```
from rttlib import dscanapilib

dscanapilib.EnableBusStatistics(ChannelHandle, Enable)
```

Purpose

To enable or disable the bus statistic frames in the receive queue.

Description

The CAN messages containing the bus statistics can be read from the receive queue if bus statistics are enabled for a channel. CAN messages containing bus statistics are identified by the MessageType flag.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel for which to enable/disable bus statistic frames.
Enable	Boolean	Specifies whether the receiving of bus statistic frames in the receive queue is enabled or disabled. ▪ True: Enable bus statistic frames in receive queue. ▪ False: Disable bus statistic frames in receive queue.

Return value

–

Related topics

References

Class Description (dscanapilib)..113
EncodeBusStatistics...117

EncodeBusStatistics

Class

dscanapilib

Syntax

```
from rttlib import dscanapilib

dscanapilib.EncodeBusStatistics(BusStatisticsCanMsg)
```

Purpose

To encode CAN bus statistics values of the CAN messages in the receive queue.

Description

The CAN messages containing the bus statistics can be read from the receive queue and then encoded to actual BusStatistics objects.

Parameters

The method uses the following parameters:

Parameter	Type	Description
BusStatisticsCanMsg	CanMessage[1]	CAN message from receive queue with MessageType `dscanapilib.mtBUSSTATISTICS`.

[1] Refer to Class Description (CanMessage) on page 138.

Return value

The method returns the following parameter:

Type	Description
BusStatistics[1]	BusStatistics object containing the statistics for a time period of a CAN channel.

[1] Refer to Class Description (BusStatistics) on page 136.

Related topics

References

Class Description (dscanapilib)..113
EnableBusStatistics..116

FlushReceiveQueue

Class

dscanapilib

Syntax

```
dscanapilib.FlushReceiveQueue(ChannelHandle)
```

Purpose

To flush the receive queue of the specified CAN channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to which the receive queue to be flushed belongs.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during flushing, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

FlushTransmitQueue

Class

dscanapilib

Syntax

```
dscanapilib.FlushTransmitQueue(ChannelHandle)
```

Purpose

To flush the transmit queue of the specified CAN channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to which the transmit queue to be flushed belongs.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during flushing, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

GetAvailableChannels

Class

dscanapilib

Syntax

```
dscanapilib.GetAvailableChannels()
```

Purpose

To get information about the available CAN channels.

Parameters

-

Return value

The method returns the following parameter:

Type	Description
List	List of `ChannelInfo` objects[1]. An `ChannelInfo` object in the list represents an available channel.

[1] Refer to ChannelInfo on page 139.

Related topics

References

Class Description (dscanapilib)... 113

GetBaudrate

Class

dscanapilib

Syntax

```
BaudRate = dscanapilib.GetBaudRate(ChannelHandle)
```

Purpose

To get the baud rate for a CAN channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel from which the baud rate is read.

Return value

The method returns the following parameter:

Type	Description
Tuple	Tuple consisting of: ▪ Clock frequency (Integer type) ▪ Bit timing parameters for non-FD case (BitTimingParameters type[1]) ▪ FD (Boolean type) ▪ Bit timing parameters FD for FD case, (BitTimingParameters type[1])

[1] Refer to BitTimingParameters on page 134.

Related topics

References

Class Description (dscanapilib).. 113

GetBusInfo

Class

dscanapilib

Syntax

```
BusInfo = dscanapilib.GetBusInfo(ChannelHandle)
```

Purpose

To get information of the bus of a CAN channel.

Description

The following information is provided:

- Bus status
- Receive error counter
- Transmit error counter
- Bus load

The method returns a `BusInfo` object that contains the information, refer to Class Description (BusInfo) on page 135.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel from which to get the bus info.

Return value

The method returns the following parameter:

Type	Description
BusInfo[1]	Object containing the requested information about the bus.

[1] Refer to BusInfo on page 135.

Related topics

References

Class Description (dscanapilib)..113

GetBusType

Class

dscanapilib

Syntax

```
BusType = dscanapilib.GetBusType(ChannelHandle)
```

Purpose

To get the bus type for a CAN channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel from which to get the bus type.

Return value

The method returns the following parameter:

Type	Description
String	Bus type as string.

Related topics

References

Class Description (dscanapilib)..113

GetErrorText

Class

dscanapilib

Syntax

```
ErrorString = dscanapilib.GetErrorText(ErrorCode)
```

Purpose

To get the error text that is related to a specified error code.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ErrorCode	Integer	Error code to translate into an error text.

Return value

The method returns the following parameter:

Type	Description
String	Error message that is related to the specified error code.

Related topics

References

Class Description (dscanapilib)..113

GetHardwareTime

Class

dscanapilib

Syntax

```
HardwareTime = dscanapilib.GetHardwareTime(ChannelHandle)
```

Purpose

To get the hardware time of a specific CAN channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the CAN channel from which to get the hardware time.

Return value

The method returns the following parameter:

Type	Description
Integer	Hardware time of the CAN channel.

Related topics

References

Class Description (dscanapilib)..113

GetHardwareTimeResolution

Class

dscanapilib

Syntax

```
MyHardwareTimeResolution =
dscanapilib.GetHardwareTimeResolution(ChannelHandle)
```

Purpose

To get the hardware time resolution of a specific channel.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Handle of the channel from which to get the hardware time resolution.

Return value

The method returns the following parameter:

Type	Description
Integer	Hardware time resolution of the channel.

Related topics

References

Class Description (dscanapilib)..113

InitChannel

Class

dscanapilib

Syntax

```
AccessPermission = None
while(AccessPermission == None):
    AccessPermission = dscanapilib.InitChannel(ChannelHandle, \
        IdentifierType, RxQueueSize, FD)
    yield None
```

Purpose

To initialize a specific channel and put it in a usable state.

Description

The initialization of a CAN channel may require several model steps. You must therefore call the `InitChannel` method in every model step until the return value is `True` or `False`. As long as the initialization is not completed, the method returns `None`.

Tip

You can implement a timeout for the repetitive call to `dscanapilib.InitChannel()` to prevent an endless loop or a stalling RTT sequence if the initialization do not succeed or throw an exception.

```
TimeOutSteps = 20
AccessPermission = None
while(AccessPermission == None):
    AccessPermission = dscanapilib.InitChannel(ChannelHandle, \
        dscanapilib.ctSTDXTD, 10, True)
    TimeOutSteps -= 1
    if(TimeOutSteps <= 0):
        raise Exception("Could not initialize CAN channels.")
    yield None
yield None
```

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be initialized.
IdentifierType	Integer	Type of CAN identifier to be used for message reception. Valid Types: ▪ `1: dscanapilib.ctSTD` Standard identifier ▪ `2: dscanapilib.ctXTD` Extended identifier ▪ `3: dscanapilib.ctSTDXTD` Standard and extended identifier
RxQueueSize	Integer	Specifies how many CAN messages can be stored in the receive queue.
FD	Boolean	Specifies whether FD is enabled if supported. ▪ `True`: CAN FD is enabled. ▪ `False`: CAN FD is disabled.

Return value

The method returns the following parameter:

Type	Description
None	The initialization is not completed.
Boolean	The initialization is successfully completed. It indicates whether you have access permission for the specified CAN channel. ▪ `True`: Access permission given. ▪ `False`: No access permission. A `False` return value does not mean that it is not possible or permitted to send or receive messages. It indicates that the channel's baud rate and other properties cannot be changed.

Related topics

References

Class Description (dscanapilib)........113

ReadReceiveQueue

Class

dscanapilib

Syntax

```
RxMessageList = dscanapilib.ReadReceiveQueue(ChannelHandle)
```

Purpose

To read the CAN messages from the receive queue of a CAN channel.

Description

The `ReadReceiveQueue` method reads the received CAN messages from the receive buffer of the specified CAN channel and returns them via a list.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be read.

Return value

The method returns the following parameter:

Type	Description
List	List of CanMessage objects[1].

[1] Refer to CanMessage on page 137.

Related topics

Basics

Receiving CAN Messages with the rttlib.dscanapilib Module (Real-Time Testing Guide 🕮)

RegisterChannel

Class

dscanapilib

Syntax

```
ChannelHandle = dscanapilib.RegisterChannel(VendorName,
InterfaceName, InterfaceSerialNumber, ChannelIdentifier)
```

Purpose

To register a CAN channel and get a handle for subsequent function calls.

Description

Before you can use a CAN channel, you must register it. To register it, you must specify the CAN interface type, CAN interface index, and the controller index of the desired channel.

> **Tip**
>
> To get the serial numbers, user strings, and indices of the CAN interfaces, use the `GetAvailableChannels` method.

Note

When a registered CAN channel is not required any longer, you must unregister it via the `UnregisterChannel` method. If a CAN channel remains registered, the dependencies in the driver cannot be cleared so that subsequent calls might fail or you might not get access permission.

Parameters

The method uses the following parameters:

Parameter	Type	Description
VendorName	String	Vendor name of the CAN interface.
InterfaceName	String	Interface name of the CAN interface.
InterfaceSerialNumber	String	Interface serial number of the CAN interface.
ChannelIdentifier	String	Channel identifier of the CAN interface.

Return value

The method returns the following parameter:

Type	Description
Integer	Channel handle of successfully registered CAN interface.

Related topics

HowTos

How to Prepare a CAN Channel for Using it with the rttlib.dscanapilib Module (Real-Time Testing Guide 🕮)

References

GetAvailableChannels........ 120
UnregisterChannel........ 133

ResetHardwareTime

Class

dscanapilib

Syntax

```
dscanapilib.ResetHardwareTime(ChannelHandle)
```

Purpose

To reset the hardware time of a CAN interface.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be reset.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during reseting the hardware time, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

SetAcceptance

Class

dscanapilib

Syntax

```
dscanapilib.SetAcceptance(ChannelHandle, StandardCanIdentifiersCode,
StandardCanIdentifiersMask, ExtendedCanIdentifiersCode,
ExtendedCanIdentifiersMask)
```

Purpose

To specify the acceptance for a CAN channel to filter incoming CAN messages by their identifiers.

Description

The method behaves like the `SetAcceptance` function of the dSPACE CAN API, refer to DSCAN_SetAcceptance (dSPACE CAN API 2.0 C Reference).

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be configured.
StandardCanIdentifiersCode	Integer	Specify the acceptance code for standard identifiers.
StandardCanIdentifiersMask	Integer	Specify the acceptance mask for standard identifiers.

Parameter	Type	Description
ExtendedCanIdentifiersCode	Integer	Specify the acceptance code for extended identifiers.
ExtendedCanIdentifiersMask	Integer	Specify the acceptance mask for extended identifiers.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during setting, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

SetBaudrate

Class

dscanapilib

Syntax

```
dscanapilib.SetBaudrate(ChannelHandle, ClockFrequency,
BitTimingParameters, BitTimingParametersFd)
```

Purpose

To set the baud rate for a channel.

Description

The method behaves like the `SetBaudrate` function of the dSPACE CAN API, refer to DSCAN_SetBaudrate (dSPACE CAN API 2.0 C Reference).

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be configured.
ClockFrequency	Integer	Frequency of the clock of the CAN interface.
BitTimingParameters	BitTimingParameters[1)]	Bit timing parameters for CAN traffic.
BitTimingParametersFd	BitTimingParameters[1)]	Bit timing parameters for CAN FD traffic.

[1)] Refer to BitTimingParameters on page 134.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during setting, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

SetChannelOutput

Class

dscanapilib

Syntax

```
dscanapilib.setChannelOutput(ChannelHandle, Mode)
```

Purpose

To specify whether a CAN channel operates in normal or silent mode.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be configured.
Mode	Boolean	The desired output mode: ▪ `True`: Silent mode (acknowledges are not generated) ▪ `False`: Normal mode (acknowledges are generated)

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during setting, `None` is returned.

Related topics

References

Class Description (dscanapilib)..113

SetTransmitAcknowledge

Class

dscanapilib

Syntax

```
dscanapilib.SetTransmitAcknowledge(ChannelHandle, AcknowledgeState)
```

Purpose

To activate or deactivate the transmit acknowledge for a CAN channel.

Description

If the transmit acknowledge is active, the transmitting CAN controller generates an acknowledge message when the CAN messages have successfully received by another CAN bus member.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be configured.
AcknowledgeState	Boolean	Activates or deactivates the transmit acknowledge state: ▪ `True`: Activates the transmit acknowledge (default). ▪ `False`: Deactivates the transmit acknowledge.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during setting, `None` is returned.

Related topics

References

Class Description (dscanapilib).. 113

TransmitMessages

Class

dscanapilib

Syntax

```
dscanapilib.TransmitMessages(ChannelHandle, TxCanMessages)
```

Purpose

To copy CAN messages to the send buffer for transmission.

Description

To transmit CAN messages, they are copied to the send buffer of a CAN channel. The send buffer is a first-in-first-out buffer, so the CAN messages are sent in the order you copied them to the buffer.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be used for transmission.
TxCanMessages	List	List of `CanMessage` objects[1] to be sent.

[1] Refer to CanMessage on page 137.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during setting, `None` is returned.

Related topics

Basics

Sending CAN Messages with the rttlib.dscanapilib Module (Real-Time Testing Guide 🕮)

UnregisterChannel

Class

dscanapilib

Syntax

```
dscanapilib.UnregisterChannel(ChannelHandle)
```

Purpose

To unregister a CAN channel that is currently in use.

Description

This method frees all dependencies of the selected CAN channel. The channel handle becomes invalid.

You can unregister a CAN channel only if it was registered via the `RegisterChannel` method. When a registered CAN channel is not required any longer, you must unregister it. If a CAN channel remains registered, the dependencies in the driver cannot be cleared so that subsequent calls might fail or you might not get access permission.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ChannelHandle	Integer	Channel handle of the channel to be unregistered.

Return value

The method returns the following parameter:

Type	Description
None	If no errors occur during unregistering, None is returned.

Related topics

References

RegisterChannel..127

BitTimingParameters

Purpose

To get/set all the necessary parameters for the baud rate.

Class Description (BitTimingParameters)

Syntax

```
MyBitTimingParameters = dscanapilib.BitTimingParameters()
```

Purpose

To get/set all the necessary parameters for the baud rate.

Description

For information on the attributes for specifying the baud rate, refer to Basics on Bit Timing Parameters and Baud Rates (dSPACE CAN API 2.0 C Reference 📖).

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
SJW	Integer	To get/set the synchronization jump width value.
BRP	Integer	To get/set the baud rate prescaler.
SAM	Integer	To get/set the sample mode: ▪ 0: One sample (high-speed buses) ▪ 1: Three samples (low/medium-speed buses)
TSEG1	Integer	To get/set the bit time segment 1.
TSEG2	Integer	To get/set the bit time segment 2.

Methods

–

Related topics

Basics

Handling CAN Messages Using the rttlib.dscanapilib Module (Real-Time Testing Guide 📖)

BusInfo

Purpose

To get and summarize information about the bus.

Class Description (BusInfo)

Syntax

```
MyBusInfo = dscanapilib.GetBusInfo(MyChannelHandle)
```

Purpose

To get and summarize information about the bus.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
BusStatus	Integer	To get the bus status: ▪ 0: `dscanapilib.UNKNOWN` ▪ 1: `dscanapilib.bsACTIVE` ▪ 2: `dscanapilib.bsPASSIVE`

Attributes	Type	Purpose
		▪ 3: `dscanapilib.bsWARNING` ▪ 4: `dscanapilib.bsBUSOFF`
RxErrorCounter	Integer	To get the receive error counter.
TxErrorCounter	Integer	To get the transmit error counter.
BusLoad	Integer	To get the bus load in percent.

Methods

-

Related topics

References

GetBusInfo.. 121

BusStatistics

Purpose

To hold all bus statistics information of a channel for a specified time.

Class Description (BusStatistics)

Syntax

```
EncodedMsg = dscanapilib.EncodeBusStatistics(BusStatisticsCanMsg)
```

Purpose

To hold all bus statistics information of a channel for a specified time.

Description

An object of this class is created using the `dscanapilib.EncodeBusStatistics()` function with a `CanMessage` object which has the MessageType `dscanapilib.mtBUSSTATISTICS`. You can use the resulting `BusStatistics` object to analyze the traffic on the CAN bus.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
Flags	Integer	Flag indicating which statistics information this `BusStatistics` object contains: ▪ `dscanapilib.bfERRORFRAMES`

Attributes	Type	Purpose
		▪ `dscanapilib.bfRXSTDFRAMES` ▪ `dscanapilib.bfTXSTDFRAMES` ▪ `dscanapilib.bfRXXTDFRAMES` ▪ `dscanapilib.bfTXXTDFRAMES` ▪ `dscanapilib.bfRXFDSTDFRAMES` ▪ `dscanapilib.bfTXFDSTDFRAMES` ▪ `dscanapilib.bfRXFDXTDFRAMES` ▪ `dscanapilib.bfTXFDXTDFRAMES`
ErrorFrames	Integer	Number of error frames on the bus.
RxStdFrames	Integer	Number of received CAN messages with a standard identifier on the bus.
TxStdFrames	Integer	Number of transmitted CAN messages with a standard identifier on the bus.
RxExtFrames	Integer	Number of received CAN messages with an extended identifier on the bus.
TxExtFrames	Integer	Number of transmitted CAN messages with an extended identifier on the bus.
RxStdFDFrames	Integer	Number of received CAN FD messages with a standard identifier on the bus.
TxStdFDFrames	Integer	Number of transmitted CAN FD messages with a standard identifier on the bus.
RxExtFDFrames	Integer	Number of received CAN FD messages with an extended identifier on the bus.
TxExtFDFrames	Integer	Number of transmitted CAN FD messages with an extended identifier on the bus.

Methods

–

Related topics

References

EncodeBusStatistics.. 117

CanMessage

Purpose

To create CAN messages to be sent or to read received messages.

Class Description (CanMessage)

Syntax

```
MyCanMessage = dscanapilib.CanMessage()
```

Purpose

To create CAN messages to be sent or to read received messages.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
BusInfo	BusInfo[1]	To get information about the bus. Only available when the CAN message is a bus info frame.
CanIdentifier	Integer	To get/set the CAN identifier. The length is 11 bit for a CAN message in standard identifier type and 29 bit for a CAN message in extended identifier type.
CanIdentifierType	Integer	To get/set the identifier type: ▪ 1: `dscanapilib.ctSTD`: Standard identifier ▪ 2: `dscanapilib.ctXTD`: Extended identifier
Data	List of bytes	List of bytes which represents data to be sent or received. Only available when the message is a data frame. The length of the list is specified by the DLC.
DLC	Integer	To get/set the number of bytes of the CAN message (DLC: data length code) For CAN messages, the maximum data length is 8 bytes. For CAN FD messages, the maximum data length is 64 bytes. Only the following DLC values are valid: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Flags	Integer	To get/set additional information about a CAN message. Transmit flags: ▪ `dscanapilib.mfTXFD` ▪ `dscanapilib.mfTXFDBRS` Receive flags ▪ `dscanapilib.mfRXFD` ▪ `dscanapilib.mfRXFDBRS` ▪ `dscanapilib.mfRXTXACK` ▪ `dscanapilib.mfRXBUFOVERRUN` ▪ `dscanapilib.mfRXHWBUFOVERRUN` ▪ `dscanapilib.mfRXERRSTATEINDICATOR`
MessageType	Integer	To get/set the message type ▪ 1: `dscanapilib.mtDATA`: Data frame ▪ 2: `dscanapilib.mtREMOTE`: Remote frame ▪ 3: `dscanapilib.mtERROR`: Error frame ▪ 4: `dscanapilib.mtBUSINFO`: Bus info frame ▪ 5: `dscanapilib.mtBUSSTATISTICS`: Bus statistics frame
Timestamp	Integer	To get the time stamp of the CAN message.

[1] Refer to BusInfo on page 135.

Methods

–

Related topics

Basics

Handling CAN Messages Using the rttlib.dscanapilib Module (Real-Time Testing Guide 🕮)

ChannelInfo

Purpose

To get information about the available CAN channels.

Class Description (ChannelInfo)

Syntax

```
MyChannelInfos = dscanapilib.GetAvailableChannels()
MyChannelInfo = MyChannelInfos[0]
```

Purpose

To get information about the available CAN channels.

Attributes

The class contains the following attributes:

Attributes	Type	Purpose
ChannelCapabilities	Integer	To get the information whether the channel supports CAN FD ▪ 1: `dscanapilib.ccFD`: The channel supports CAN FD
ChannelIdentifier	String	To get the identifier of the channel.
InterfaceName	String	To get the name of the interface.
InterfaceSerialNumber	String	To get the serial number of the CAN interface.
VendorName	String	To get the name of the vendor.

Methods	–
Related topics	References GetAvailableChannels......120

rttlib.datastream Module

Introduction

This module provides a class for streaming data from MAT files and MDF4 files on the host PC to an RTT sequence to stimulate variables.

Where to go from here

Information in this section

CreateVariableMap..141
To create a variable map for the variables of the MAT file.

VariableMap..143
To map MAT file data to variable objects for data streaming.

MatFile..145
To get an object for data streaming from a MAT file.

CreateVariableMapMDF..150
To create a variable map for the channels of the MDF file.

VariableMapMDF..152
To map MDF file data to variable objects for data streaming.

MDFFile..154
To get an object for data streaming from an MDF file.

Information in other sections

Basics of Data Replay Using MAT Files (Real-Time Testing Guide 📖)
You can stimulate variable objects in an RTT sequence by data replay of MAT file variables.

Basics of Data Replay Using ASAM MDF (MF4) Files (Real-Time Testing Guide 📖)
You can stimulate variable objects in an RTT sequence by data replaying ASAM MDF file variables.

General Limitations for Real-Time Testing (Real-Time Testing Guide 📖)
Some limitations apply for Real-Time Testing.

CreateVariableMap

Purpose

To create a variable map for the variables of the MAT file.

CreateVariableMap Class Description

Syntax

```
from rttlib import datastream
VariableMap = datastream.CreateVariableMap("Time")
```

Purpose

To create a variable map for the variables of the MAT file.

Description

A variable map object is the mapping of MAT file data to variable or dynamic variable objects and needed as input for data streaming. Refer to MatFile Class Description on page 146. Each variable map refers to a unique time vector, whose name is passed to the constructor of the variable map object. Then a variable object and its associated MAT file vector name can be added to the map object. Refer to AddVariable Method on page 144.

The mapping is designed to be independent of any specific MAT file. The MAT file used for data streaming must include the vector names used in the mapping. The data type of the MAT file vectors must be double.

The variable map must only be created during the initialization of the RTT sequence. It must be completed before you create a datastream object for it. Refer to MatFile Class Description on page 146.

Parameter

The class uses the following parameter:

Parameter	Type	Description
TimeVariableName	string	Name of time vector in the MAT file. For example: "Time".

Return value

The class returns a value of the following type:

Type	Description
VariableMap[1]	Variable map object.

[1] Refer to VariableMap Class Description on page 143.

Example

The following example shows how to use the method:

```
from rttlib import datastream
# Map variable objects to the MAT File variables
# The constructor uses the MAT file time variable name
VariablesToStimulate = datastream.CreateVariableMap("Time")
```

Related topics

Basics

Basics of Data Replay Using MAT Files (Real-Time Testing Guide 📖)
VariableMap.. 143

VariableMap

Purpose

To map MAT file data to variable objects for data streaming.

Where to go from here

Information in this section

VariableMap Class Description.. 143
To describe the class and its attributes.

AddVariable Method.. 144
To add variables stored in the MAT file to the variable map for a RTT sequence.

VariableMap Class Description

Syntax

```
from rttlib import datastream
VariableMap = datastream.CreateVariableMap("Time")
```

Purpose

To map MAT file data to variable objects for data streaming.

Description

Before you can add variables to a variable map object, the object must be created using the `CreateVariableMap` method. Refer to . A MAT file vector/variable object pair can be added to existing variable maps. The vectors of the MAT file must be of data type 'double'. The variable objects are stimulated with the data content of the associated MAT file vector at the time stamps of the variable map time vector. Only mapped variables are stimulated.

The variable mapping must be created in the sequence's init phase and completed before the variable map object is used for creating a MatFile object.

Methods

The following method is part of the class:

Method	Purpose
AddVariable	To map a variable or dynamic variable object to a MAT file vector and add them to the variable map for data streaming. Refer to AddVariable Method on page 144.

Related topics

Basics

Data Replay in RTT Sequences (Real-Time Testing Guide 🕮)

References

CreateVariableMap Class Description... 142

AddVariable Method

Class

VariableMap

Syntax

```
from rttlib import datastream
VariableMap.AddVariable(VariableNameMATFile, RTTVariable)
```

Purpose

To map a variable object to a MAT file vector and add them to the variable map for data streaming.

Description

The method adds a MAT file vector/variable object pair to the variable map for data streaming.

Parameter

The method uses the following parameters:

Parameter	Type	Description
VariableNameMATFile	String	Name of the variable in the MAT file (source), for example, "Signal_1".
RTTVariable	Object	Variable object

Return value

–

Example

The following example shows how to use the method:

```python
from rttlib import variable
from rttlib import datastream
# Module-global variables
# Create variable objects for accessing Simulink signals.
Frequency      = variable.Variable(r'Model Root/x disp/Frequency')
SpringConstant = variable.Variable(r'Model Root/Model Parameters/C/Value')
Mass           = variable.Variable(r'Model Root/Model Parameters/m/Value')
Damper         = variable.Variable(r'Model Root/Model Parameters/d/Value')
# Map variable objects to the MAT File variables.
# The constructor uses the MAT file time variable name.
VariablesToStimulate = datastream.CreateVariableMap("Time")
# All variables objects must be mapped to a MAT file variable name.
VariablesToStimulate.AddVariable("Signal_1", Frequency)
VariablesToStimulate.AddVariable("Signal_2", SpringConstant)
VariablesToStimulate.AddVariable("Signal_3", Mass)
VariablesToStimulate.AddVariable("Signal_4", Damper)
...
```

Related topics

Basics

Basics of Data Replay Using MAT Files (Real-Time Testing Guide 🕮)
Read/Write Access to Variables of the Simulation Application (Real-Time Testing Guide 🕮)

MatFile

Purpose

To get an object for data streaming from a MAT file.

Where to go from here

Information in this section

MatFile Class Description 146
To describe the class and its attributes.

Replay Method 149
To start data streaming of MAT file data in the MainGenerator function.

MatFile Class Description

Syntax

```
from rttlib import datastream
Datastream = datastream.MatFile(MatFileName, VariableMap)
```

Purpose

To get an object for data streaming from a MAT file.

Description

The class creates a datastream object which you can use in the MainGenerator function to stream data. Refer to Replay Method on page 149. Before you can use the method, you must have created a variable map. Refer to CreateVariableMap Class Description on page 142 and AddVariable Method on page 144. You must complete the variable map before creating the datastream object.

The MAT file is usually not completely loaded to the real-time hardware. When data replay starts, data values are reloaded and can be replayed in real time.

You must create the datastream object during the RTT sequence's initialization phase. It is not possible to call the `datastream.MatFile()` method in MainGenerator function of the RTT sequence, because setting up the whole system on the real-time platform and on the host PC requires computation time. The MainGenerator function works under real-time conditions, so the new data is likely replayed within milliseconds. This is too fast to be synchronized with the host.

The datastream object must not be deleted if the corresponding iterator object is used.

Note

The MAT file you use must fulfill the following preconditions:
- The MAT file must contain at least two one-dimensional arrays. One array must contain monotonically increasing values for the time axis (x-axis).
- The data must be of 'double' type.
- The MAT file can be used only if it does not contain a substructure.

Parameters

The following parameters are part of the class:

Note

Some of the parameters are optional. If you want to set optional parameters, you must specify their names when creating the object. Otherwise, the values cannot be assigned to the parameters correctly. For example, to set the replay mode, use:

```
myStream = datastream.MatFile(MatFileName, VariableMap,\
                          ReplayMode = datastream.RM_BACKWARD)
```

Parameter	Type	Description
MatFileName	String	The name of the file whose data values are streamed. It must contain the full path to the file on the host PC. The file must be stored on the host PC, for example, `r"C:\Tests\MyDataStream.mat"`. The data values in the file must be of 'double' data type. MAT files can be used if they do not contain a substructure.
VariableMap	Object	The variable map with all variables to be streamed. Refer to CreateVariableMap Class Description on page 142 and AddVariable Method on page 144.
BufferSize	Integer	(Optional) The size of the buffer which is used for data streaming, see below. The parameter is only evaluated for DS1006 and MicroAutoBox II. For all other platforms: The buffer size is fixed and cannot be modified. The parameter is not evaluated and exists only for compatibility and portability reasons.
ReplayMode	Integer	(Optional) The mode for data replay. Four modes are implemented: `RM_STRICT`, `RM_SAMPLED`, `RM_LINEAR`, and `RM_BACKWARD`. The `RM_STRICT` mode is used by default. Refer to Replay Mode (Real-Time Testing Guide 🕮).

BufferSize Using the optional `BufferSize` parameter, you can change the automatically configured buffer size (the default buffer size is configured to buffer data for 100 ms). The buffer size affects the execution time required for data streaming. Usually, it is not necessary to change the value. If the buffer size is too small, for example, for a slow connection or a slow host computer, data streaming can abort. Choosing a very large buffer will lengthen the sequence's initialization phase and thus trigger a timeout while creating the RTT sequence. The following formula shows how you can approximate the required buffer size:

`BufferSize = Number_of_variables · Data_type_size · Data_rate`

where

`Number_of_variables` is the number of variables that are used for data streaming (signals and time vector)

`Data_type_size` is the size of the data type in bytes

`Data_rate` is the time to be buffered/size of sampling step

Example: If you want to stream 51 variables (50 signals + time vector) with a data size of 8 bytes and a resolution of the time vector of 0.01 s for a time buffer of 0.1 s, a buffer size of 4080 is required (= 51 · 8 · (0.1/0.01)).

The calculated value is internally multiplied by 8.

Methods

The following method is part of the class:

Method	Purpose
Replay	To start data streaming of MAT file data. Refer to Replay Method on page 149.

Return value

The class returns a value of the following type:

Type	Description
data stream object	Data stream object used for streaming data

Example

The following example shows how to use the class:

```python
from rttlib import variable
from rttlib import datastream
# Module global variables
# Create variable objects for accessing Simulink signals
WarningLightSwitch = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
TurnSignalLeft     = variable.Variable(r'Model Root/FrontLightEcu/TurnSignalLeft')
BatteryVoltage     = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
TurnSignalLever    = variable.Variable(r'Model Root/TurnSignalLever[-1..1]/Value')

matFileName    = 'MyMatFile.mat'
# Map variable objects to the MAT file variables
# The constructor uses the MAT file time variable name
variablesToStimulate = datastream.CreateVariableMap("Time")
# All variables objects must be mapped to a MAT file variable name.
variablesToStimulate.AddVariable("Signal_1", WarningLightSwitch)
variablesToStimulate.AddVariable("Signal_2", TurnSignalLeft)
variablesToStimulate.AddVariable("Signal_3", BatteryVoltage)
variablesToStimulate.AddVariable("Signal_4", TurnSignalLever)
# Create a data stream
myStream = datastream.MatFile(matFileName, variablesToStimulate, \
              ReplayMode = datastream.RM_STRICT)
```

Related topics

Basics

Basics of Data Replay Using MAT Files (Real-Time Testing Guide 📖)

Replay Method

Class

MatFile

Syntax

```
from rttlib import datastream
RTTVariable = variable.Variable(r'Model Root/Variable')
VariableMap = datastream.CreateVariableMap("Time")
VariableMap.AddVariable("Signal_1", RTTVariable)
DataStream = datastream.MatFile(r'C:\DataStream.mat', VariableMap)
def MainGenerator():
   yield DataStream.Replay()
```

Purpose

To start data streaming of MAT file data.

Description

Before you can use the `Replay` method, an object for data streaming must exist. Refer to MatFile Class Description on page 146. When you use the method, data streaming from the host PC to the real-time platform starts.

As the `Replay` method is a generator function, it must be prefixed with the `yield` statement.

The replay starts in the execution step in which the generator function is called for the first time and ends after the generator finished. There are several ways for the generator to finish:

- The MAT file has been replayed completely.
- The generator is terminated by `scheduler.ParallelRace()`.
- Shutting down of the Real-Time Test Manager Server stops data replay.

The data of the MAT file vectors is replayed without any modifications. This includes the following:

- The variable value is set at the time given by the variable map time vector with a value given by the mapped data vector in the default mode RM_STRICT. Refer to *ReplayMode* in MatFile Class Description on page 146.
- There is no normalization of the time vector to zero. This means if the first entry in the MAT file time vector is '5.5', for example, the first value is stimulated 5.5 seconds after the start of the replay generator.

- A MAT file time vector resolution higher than the model step size an exception occurs in the default mode RM_STRICT. Refer to `ReplayMode` in MatFile Class Description on page 146.

The replay of a MAT file can be restarted by calling the `Replay` method again or by starting the RTT sequence again. An instance of a `datastream.MatFile` object cannot be started by the `Replay` method twice in parallel. If a datastream object is deleted (for example, using `del()` or assigning `None`), the replay cannot be restarted. When RTT sequences are removed, the created datastream objects are deleted.

Parameter

–

Return value

–

Example

The example shows how to use the method:

```
MyStream = datastream.MatFile(MatFileName, VariablesToStimulate)
...
def MainGenerator():
   # Start the data replay.
   yield MyStream.Replay()
   yield None
   # Start the data replay again.
   yield MyStream.Replay()
```

Related topics

Basics

Basics of Data Replay Using MAT Files (Real-Time Testing Guide 📖)

CreateVariableMapMDF

Purpose

To create a variable map for the channels of the MDF file.

CreateVariableMapMDF Class Description

Syntax

```
from rttlib import datastream
VariableMap = datastream.CreateVariableMapMDF("GroupNameString", \
    "GroupSourceString", "GroupPathString")
```

Purpose

To create a variable map for the channels of the MDF file.

Description

A variable map object is the mapping of ASAM MDF file data to variable or dynamic variable objects and is needed as input for data streaming. Each variable map refers to a group name, a group source, and a group path whose names are passed to the constructor of the variable map object. Then a variable object and its associated name, group and path of the channel of the ASAM MDF file can be added to the map object. Refer to AddVariable Method on page 153.

The variable map must be created only during the initialization of the RTT sequence. It must be completed before you create a datastream object for it. Refer to MDFFile Class Description on page 154.

Parameter

The class uses the following parameter:

Parameter	Type	Description
GroupName	String	Group name of the group used in this variable map.
GroupSource	String	Group source of the group used in this variable map.
GroupPath	String	Group path of the group used in this variable map.

Return value

The class returns a value of the following type:

Type	Description
VariableMapMDF[1]	Variable map object.

[1] Refer to VariableMapMDF Class Description on page 152.

Example

The following example shows how to use the method:

```
from rttlib import datastream
# Map variable objects to the MDF File variables.
# The constructor uses the group name, groups source, and group path.
VariablesToStimulate = datastream.CreateVariableMapMDF("Base Task", \
    "Turnlamp", "Demo Signals")
```

Related topics

Basics

Basics of Data Replay Using ASAM MDF (MF4) Files (Real-Time Testing Guide 📖)

References

MDFFile Class Description 154

VariableMapMDF

Purpose

To map MDF file data to variable objects for data streaming.

Where to go from here

Information in this section

VariableMapMDF Class Description..152
To describe the class and its attributes.

AddVariable Method...153
To map a variable object to an MDF file channel and add them to the variable map for data streaming.

VariableMapMDF Class Description

Syntax

```
from rttlib import datastream
VariableMapMDF = datastream.CreateVariableMapMDF(GroupName, GroupSource, GroupPath)
```

Purpose

To map MDF file data to variable objects for data streaming.

Description

Before you can add variables to a variable map object, the object must be created using the `CreateVariableMapMDF` method. An MDF file channel/variable object pair can be added to existing variable maps. The variable objects are stimulated with the data content of the associated MDF file channel at the time stamps of the variable map MDF group's master channel. Only mapped variables are stimulated.

The variable mapping must be created in the sequence's init phase and completed before the variable map object is used for creating a MDFFile object.

Methods

The following method is part of the class:

Method	Purpose
AddVariable	To map a variable or dynamic variable object to an MDF file channel and add them to the variable map for data streaming. Refer to AddVariable Method on page 153.

Related topics

References

CreateVariableMapMDF Class Description.. 150

AddVariable Method

Class

VariableMapMDF

Syntax

```
from rttlib import datastream
VariableMap.AddVariable(ChannelNameMDFFile, RTTVariable, \
    ChannelSourceMDFFile, ChannelPathMDFFile)
```

Purpose

To map a variable object to an MDF file channel and add them to the variable map for data streaming.

Description

The method adds an MDF file channel object to the variable map for data streaming.

Parameter

The method uses the following parameters:

Parameter	Type	Description
ChannelNameMDFFile	String	Channel name of the channel used in this variable map.
RTTVariable	Object	Variable object.
ChannelSourceMDFFile	String	Channel source of the channel used in this variable map.
ChannelPathMDFFile	String	Channel path of the channel used in this variable map.

Return value

–

Example

The following example shows how to use the method:

```
from rttlib import variable
from rttlib import datastream
```

```
# Module-global variables:
# Create variable objects for accessing Simulink signals.
WarningLightSwitch       = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
TurnSignalLeft           = variable.Variable(r'Model Root/RearLightEcu/TurnSignalLeft')
BatteryVoltage           = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
TurnSignalLever          = variable.Variable(r'Model Root/TurnSignalLever[-1..1]/Value')
# Map variable objects to the MDF File channel.
# The constructor uses the group name, group source, and group path.
VariablesToStimulate = datastream.CreateVariableMapMDF("Base Task", "Turnlamp", "Demo Signals")
# All variable objects must be mapped to an MDF file channel.
variablesToStimulate.AddVariable("Sine Wave", WarningLightSwitch, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Stairs", TurnSignalLeft, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Add Const Sine Wave", BatteryVoltage, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Ramp", TurnSignalLever, "Turnlamp", "Demo Signals")
...
```

Related topics

References

CreateVariableMapMDF Class Description 150
MDFFile Class Description 154
VariableMapMDF Class Description 152

MDFFile

Purpose

To get an object for data streaming from an MDF file.

Where to go from here

Information in this section

MDFFile Class Description 154
To describe the class and its attributes.

Replay Method 158
To start data streaming of MDF file data in the MainGenerator function.

MDFFile Class Description

Syntax

```
from rttlib import datastream
Datastream = datastream.MDFFile(mf4FileName, VariableMapMDF,
ReplayMode = datastream.RM_BACKWARD)
```

Purpose

To get an object for data streaming from an MDF file.

Description

The class creates a datastream object that you can use in the MainGenerator function to stream data. Refer to Replay Method on page 158. Before you can use the method, you must have created a variable map. Refer to CreateVariableMapMDF Class Description on page 150 and AddVariable Method on page 153. You must complete the variable map before creating the datastream object.

Using the `Start` and `Duration` attributes, you can specify an interval of the MDF file to be streamed. By default, all data values of the channels added to the variable map are replayed from start to end. However, data streaming is started in the model step in which the `Replay` method is called.

The MDF file is usually not completely loaded to the real-time hardware. When data replay starts, data values are reloaded and can be replayed in real time.

You must create the datastream object during the RTT sequence's initialization phase. It is not possible to call the `datastream.MDFFile()` method in MainGenerator function of the RTT sequence, because setting up the whole system on the real-time platform and on the host PC requires computation time. The MainGenerator function works under real-time conditions, so the new data is likely replayed within milliseconds. This is too fast to be synchronized with the host.

The datastream object must not be deleted if the corresponding iterator object is used.

Parameters

The following parameters are part of the class:

Note

Some of the parameters are optional. If you want to set optional parameters, you must specify their names when creating the object. Otherwise, the values cannot be assigned to the parameters correctly. For example, to set the replay mode, use:

```
mystream = datastream.MDFFile(mf4FileName, VariableMapMDF,\
                              ReplayMode = datastream.RM_BACKWARD)
```

Parameter	Type	Description
Mf4FileName	String	The name of the MDF file whose data values are streamed. It must contain the full path to the file on the host PC. The file must be stored on the host PC. For example: `r"C:\Tests\MyDataStream.mf4"`.
VariableMapMDF	Object	The variable map with all variables to be streamed. Refer to CreateVariableMapMDF Class Description

Parameter	Type	Description
		on page 150 and AddVariable Method on page 153.
BufferSize	Integer	(Optional) The size of the buffer that is used for data streaming, see below. The parameter is only evaluated for DS1006 and MicroAutoBox II. For all other platforms: The buffer size is fixed and cannot be modified. The parameter is not evaluated and exists only for compatibility and portability reasons.
ReplayMode	Integer	(Optional) The mode for data replay. Four modes are implemented: `RM_STRICT`, `RM_SAMPLED`, `RM_LINEAR`, and `RM_BACKWARD`. The `RM_STRICT` mode is used by default. Refer to Replay Mode (Real-Time Testing Guide 🕮).
Start	Float	(Optional) Start time in the MDF file. The MDF channel values after the start time are used for the replay. The default is `infinity`, meaning the replay starts with the very first value of the MDF channel.
Duration	Float	(Optional) Duration of the replay. The MDF channel values from the start time throughout specified duration are used for the replay. The default is `infinity`, meaning the replay ends with the last value of the MDF channel.

BufferSize Using the optional `BufferSize` parameter, you can change the automatically configured buffer size (the default buffer size is configured to buffer data for 100 ms). The buffer size affects the execution time required for data streaming. Usually, it is not necessary to change the value. If the buffer size is too small, for example, for a slow connection or a slow host computer, data streaming can abort. Choosing a very large buffer will lengthen the sequence's initialization phase and thus trigger a timeout while creating the RTT sequence. The following formula shows how you can approximate the required buffer size:

```
BufferSize = Number_of_variables · Data_type_size · Data_rate
```

where

`Number_of_variables`	is the number of variables that are used for data streaming (signals and time vector)
`Data_type_size`	is the size of the data type in bytes
`Data_rate`	is the time to be buffered/size of sampling step

Example: If you want to stream 51 variables (50 signals + time vector) with a data size of 8 bytes and a resolution of the time vector of 0.01 s for a time buffer of 0.1 s, a buffer size of 4080 is required (= 51 · 8 · (0.1/0.01)).

The calculated value is internally multiplied by 8.

Methods

The following method is part of the class:

Method	Purpose
Replay	To start data streaming of MDF file data. Refer to Replay Method on page 158.

Return value

The class returns a value of the following type:

Type	Description
datastream object	Data stream object used for streaming data

Example

The following example shows how to use the class:

```
from rttlib import variable
from rttlib import datastream
# Module-global variables:
# Create variable objects for accessing Simulink signals.
WarningLightSwitch = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
TurnSignalLeft     = variable.Variable(r'Model Root/FrontLightEcu/TurnSignalLeft')
BatteryVoltage     = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
TurnSignalLever    = variable.Variable(r'Model Root/TurnSignalLever[-1..1]/Value')

mf4FileName    = 'MyM4fFile.mf4'
# Map variable objects to the channels of the MDF file.
# The constructor uses the name, source, and path of a group in an MDF file.
variablesToStimulate = datastream.CreateVariableMapMDF("Base Task", "Turnlamp", "Demo Signals")
# All variable objects must be mapped to a channel within the group
# used for the variable map constructor.
# The method uses the name, source, and path of a channel in an MDF file.
variablesToStimulate.AddVariable("Sine Wave", WarningLightSwitch, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Stairs", TurnSignalLeft, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Add Const Sine Wave", BatteryVoltage, "Turnlamp", "Demo Signals")
variablesToStimulate.AddVariable("Ramp", TurnSignalLever, "Turnlamp", "Demo Signals")
# Create a data stream.
myStream = datastream.MDFFile(mf4FileName, variablesToStimulate, ReplayMode = datastream.RM_BACKWARD)
```

Related topics

Basics

Basics of Data Replay Using ASAM MDF (MF4) Files (Real-Time Testing Guide 🕮)

Replay Method

Class

MDFFile

Syntax

```
def MainGenerator():
   yield DataStream.Replay()
```

Purpose

To start data streaming of MDF file data.

Description

Before you can use the `Replay` method, an object for data streaming must exist. Refer to MDFFile Class Description on page 154. When you use the method, data streaming from the host PC to the real-time platform starts.

Because the `Replay` method is a generator function, it must be prefixed with the `yield` statement.

The replay starts in the execution step in which the generator function is called for the first time and ends after the generator finished. There are various ways for the generator to finish:

- The duration specified in the MDFFile object expires.
- The MDF file was replayed completely.
- The generator is terminated by `scheduler.ParallelRace()`.
- Shutting down of the Real-Time Test Manager Server stops data replay.

The data of the MDF file channels is replayed without any modifications in the default replay mode. This includes the following:

- The variable value is set at the time specified by the MDF group's master channel with a value specified by the mapped data channel in the default mode RM_STRICT. Refer to `ReplayMode` in MatFile Class Description on page 146.
- The replay starts in the time step in which the method is called, independently of the start value specified in the MDFFile object.
- If the time specified by the MDF group's master channel has a higher resolution than the model step size, an exception occurs in the default mode RM_STRICT. Refer to `ReplayMode` in MDFFile Class Description on page 154.

The replay of an MDF file can be restarted by calling the `Replay` method again or by starting the RTT sequence again. An instance of a `datastream.MDFFile` object cannot be started by the `Replay` method twice in parallel. If a datastream object is deleted (for example, using `del()` or assigning `None`), the replay cannot be restarted. When RTT sequences are removed, the created datastream objects are deleted.

Parameter

–

Return value

–

Example

The example shows how to use the method:

```
MyStream = datastream.MDFFile(mf4FileName, VariablesToStimulate)
...
def MainGenerator():
   # Start the data replay.
   yield MyStream.Replay()
   yield None
   # Start the data replay again.
   yield MyStream.Replay()
```

Related topics

Basics

Basics of Data Replay Using ASAM MDF (MF4) Files (Real-Time Testing Guide 🕮)

rttlib.dynamicvariable Module

Introduction

This module provides a class to represent an object for accessing dynamic variables from an RTT sequence and from the host PC. Dynamic variables can be created during the real-time application's run time.

Where to go from here

Information in this section

DynamicVariable Class.......160
To describe the class and its attributes.

Name Method.......161
To return the name of the dynamic variable.

DynamicVariable Class

Syntax

```
from rttlib import dynamicvariable
MyDynamicVariable = dynamicvariable.DynamicVariable(DynamicVariableName)
```

Purpose

To represent an object to access dynamic variables from an RTT sequence and from the host PC. The variables are readable and writeable both from the host PC and the simulation platform.

Description

Dynamic variables can be created during the real-time application's run time.

You can remove dynamic variables only by reloading the real-time application.

Parameter

The method uses the following parameter:

Parameter	Type	Description
DynamicVariableName	String	Name of the dynamic variable. The variable’s name is unique in the namespace of the Python interpreter where all RTT sequences are created. If the variable name does not yet exist, a new variable object is created. If the variable name already exists, the variable object is referenced to the existing variable object.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Value	Float	To read and write the value of the dynamic variable (only float data type)
DynamicValue	Boolean, integer, float, string, tuple	To read and write the value of the dynamic variable (several data types and a combination of them). Before the DynamicValue is read for the first time, it must be set initially. Otherwise an exception is triggered.

Methods

The following method is part of the class:

Method	Purpose
Name	To return the name of the dynamic variable. Refer to Name Method on page 161.

Example

Refer to Example of Using Dynamic Variables (Real-Time Testing Guide 📖).

The following listing shows the host script that reads the DynamicValue of the dynamic variable object.

Related topics

Basics

Basics on Dynamic Variables (Real-Time Testing Guide 📖)

References

Variable Class Description..83
Variables Class (Collection) Description...79

Name Method

Class

DynamicVariable

Syntax

```
OBJ.Name()
```

Purpose

To return the name of the dynamic variable.

Parameter

–

Return value

The method returns a value of the following type:

Type	Description
String	Name of the dynamic variable

Related topics

References

DynamicVariable Class.. 160

rttlib.errors Module

Introduction

This module handles exceptions on the real-time platform.

RTTException

RTTException Class Description

Syntax

```
from rttlib import errors
errors.RTTException(Exception Description)
```

Purpose

To represent an exception object which handles the exceptions on the simulation platform.

Description

With the RTTException class, you can generate exceptions in an RTT sequence, for example, `raise RTTException("Invalid parameter")`.

> **Tip**
>
> It is recommended to use this class, since generating string exceptions decreases system performance and causes a warning in Python 2.5 ("raising a string exception is deprecated").

Attributes

–

Methods

–

Related topics

Basics

Implementing an Exception Handling (Real-Time Testing Guide 📖)

rttlib.dsethernetapilib Module

Introduction

This module provides functions for sending and receiving Ethernet raw frames with the RTT sequences on the real-time platform.

Where to go from here

Information in this section

AccessProviderInfo Class.. 166
To get information on the Ethernet access providers on the real-time platform.

InterfaceInfo Class.. 166
To get information of an Ethernet interface of the real-time platform.

EthRawFrameHeader Class.. 167
To get information on the header of an Ethernet raw frame.

EthRawFrame Class.. 168
To create Ethernet frames to be sent or to be read.

ActivateInterface Method.. 169
To activate an Ethernet interface.

CreateBuffer Method.. 170
To create a buffer object to be used for receiving and transmitting Ethernet frames.

DeactivateInterface Method.. 170
To deactivate an Ethernet interface.

FlushReceiveQueue Method.. 171
To flush the receive queue of an Ethernet interface.

FlushTransmitQueue Method.. 172
To flush the transmit queue of an Ethernet interface.

GetAccessProviders Method.. 172
To get all access providers of Ethernet interfaces.

GetAccessProvidersCount Method.. 173
To get the number of access providers of Ethernet interfaces.

GetAvailableInterfaces Method... 173
To get all available Ethernet interfaces.

GetAvailableInterfacesCount Method... 174
To get the number of available Ethernet interfaces.

GetInterfaceCapabilities Method.. 175
To get the capabilities of an Ethernet interface.

GetInterfaceMacAddress Method... 175
To get the MAC address of an Ethernet interface.

GetSupportedFrameTypes Method..176
To get the supported frame types of an Ethernet interface.

GetSupportedFrameTypesCount Method..177
To get the number of supported frame types of an Ethernet interface.

GetTime Method...178
To get the hardware time of an Ethernet interface.

GetTimeResolution Method..178
To get the hardware time resolution of an Ethernet interface.

InitInterface Method..179
To initialize an Ethernet interface.

IsInterfaceAccessible Method..181
To check whether an Ethernet interface is accessible.

IsInterfaceAvailable Method..181
To check whether an Ethernet interface is available.

ReadFrames Method...182
To read the Ethernet frames from the receive queue of an Ethernet interface.

RegisterInterface Method...183
To register an Ethernet interface using its interface information.

SetFilter Method...184
To set a frame filter for an Ethernet interface.

TransmitFrames Method...185
To transmit the Ethernet frames using an Ethernet interface and a transmit buffer.

UnregisterInterface Method...186
To unregister an Ethernet interface.

Information in other sections

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)
You can transmit and receive frames via Ethernet in RTT sequences. Real-Time Testing provides the dsethernetapilib module for this.

Example of Sending and Receiving Frames via Ethernet (Real-Time Testing Guide 📖)
The example demonstrates how you can send and receive frames via Ethernet.

AccessProviderInfo Class

Syntax

```
from rttlib import dsethernetapilib
MyAccessProviderInfos = dsethernetapilib.GetAccessProviders()
MyAccessProviderInfo = MyAccessProviderInfos[0]
```

Purpose

To get information on the Ethernet access providers on the real-time platform.

Attributes

The class has the following attributes:

Attribute	Type	Description
AccessProviderName	String	Name of the access provider.
ApiDllName	String	Name of the API DLL.
ApiVersion	Integer	Version of the API.
RequiredApiVersion	Integer	Required version of the API.
ApiErrorState	Integer	Error state of theAPI.

Methods

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

InterfaceInfo Class

Syntax

```
from rttlib import dsethernetapilib
MyInterfaceInfos = dsethernetapilib.GetAvailableInterfaces()
MyInterfaceInfo = MyInterfaceInfos[0]
```

Purpose

To get information of an Ethernet interface of the real-time platform.

Attributes

The class has the following attributes:

Attribute	Type	Description
AccessProviderName	String	Name of access provider.
InterfaceName	String	Name of Ethernet interface.
InterfaceSerialNumber	String	Serial number of Ethernet interface.
InterfaceIdentifier	String	Identifier of Ethernet interface.
InterfaceMacAddress	List	List of bytes representing the MAC address.
InterfaceCapabilities	Integer	Flags marking the capabilities of the Ethernet interface.

Methods

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

EthRawFrameHeader Class

Syntax

```
from rttlib import dsethernetapilib
MyEthRawFrame = dsethernetapilib.EthRawFrame()
MyEthRawFrameHeader = MyEthRawFrame.Header
```

Purpose

To get information on the header of an Ethernet raw frame.

Attributes

The class has the following attributes:

Attribute	Type	Description
FrameType	Integer	Type of Ethernet frame: ▪ 1: `dsethernetapilib.ftETHERNET`: Ethernet frame with frame check sequence ▪ 2: `dsethernetapilib.ftLOOPBACK`: Loopback frame (currently not supported) ▪ 3: `dsethernetapilib.ftETHERNETNOFCS` Ethernet frame without frame check sequence (currently not supported)

Attribute	Type	Description
Flags	Integer	Flags of Ethernet frame: ▪ `dsethernetapilib.ffPHYSICALERROR`: Physical error. ▪ `dsethernetapilib.ffINVALIDLENGTH`: Invalid frame length ▪ `dsethernetapilib.ffINVALIDFCS`: Invalid frame check sequence. ▪ `dsethernetapilib.ffRXBUFFEROVERFLOW`: Receive buffer overflow. ▪ `dsethernetapilib.ffSOURCEMAC`: Automatic assignment of source MAC address.
Timestamp	Integer	Timestamp of Ethernet frame.
ControllerTimestamp	Integer	Timestamp of Ethernet controller.
RawDataLength	Integer	Length of raw data in bytes.

Methods

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

EthRawFrame Class

Syntax

```
from rttlib import dsethernetapilib
MyEthRawFrame = dsethernetapilib.EthRawFrame()
```

Purpose

To create Ethernet frames to be sent or to be read.

Attributes

The class has the following attributes:

Attribute	Type	Description
Header	EthRawFrameHeader[1)]	Header of Ethernet frame.
HeaderLength	Integer	Length of header.
Length	Integer	Length of Ethernet frame (considering raw data length and header length).

Attribute	Type	Description
RawData	List	List of bytes representing the raw data of the Ethernet frame.

[1] Refer to EthRawFrameHeader Class on page 167.

Methods

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

ActivateInterface Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.ActivateInterface(InterfaceHandle)
```

Purpose

To activate an Ethernet interface.

Description

You must register and initialize an Ethernet interface before you can activate it.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of Ethernet interface to be activated.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

References

DeactivateInterface Method.. 170

CreateBuffer Method

Syntax

```
from rttlib import dsethernetapilib
MyBuffer = dsethernetapilib.CreateBuffer(BufferSize)
```

Purpose

To create a buffer object to be used for receiving and transmitting Ethernet frames.

Description

To read Ethernet frames from the receive queue and to transmit Ethernet frames, you must create a buffer that holds the contents of all the frames to be read or transmitted.

Parameter

The function uses the following parameter:

Parameter	Type	Description
BufferSize	Integer	Size of buffer in bytes. It must be large enough to hold the contents of all expected Ethernet frames to transmit or to read.

Return value

The function returns a value of the following type:

Type	Description
Bytes	Python bytes object representing the buffer for the Ethernet frames.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

DeactivateInterface Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.DeactivateInterface(InterfaceHandle)
```

Purpose

To deactivate an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to be deactivated.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

References

ActivateInterface Method..169

FlushReceiveQueue Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.FlushReceiveQueue(InterfaceHandle)
```

Purpose

To flush the receive queue of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface which the receive queue to be flushed belongs to.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

FlushTransmitQueue Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.FlushTransmitQueue(InterfaceHandle)
```

Purpose

To flush the transmit queue of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface which the transmit queue to be flushed belongs to.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

GetAccessProviders Method

Syntax

```
from rttlib import dsethernetapilib
MyAccessProviderInfos = dsethernetapilib.GetAccessProviders()
```

Purpose

To get all access providers of Ethernet interfaces.

Parameter

–

Return value

The function returns a value of the following type:

Type	Description
List	List of the AccessProviderInfo[1] objects representing all the available access providers.

[1] Refer to AccessProviderInfo Class on page 166.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetAccessProvidersCount Method

Syntax

```
from rttlib import dsethernetapilib
MyAccessProvidersCount = dsethernetapilib.GetAccessProvidersCount()
```

Purpose

To get the number of access providers of Ethernet interfaces.

Parameter

–

Return value

The function returns a value of the following type:

Type	Description
Integer	Number of access providers.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetAvailableInterfaces Method

Syntax

```
from rttlib import dsethernetapilib

MyInterfaceInfos = dsethernetapilib.GetAvailableInterfaces()
```

Purpose

To get all available Ethernet interfaces.

Parameter

–

Return value

The function returns a value of the following type:

Type	Description
List	List of the InterfaceInfo[1] objects representing all the available Ethernet interfaces.

[1] Refer to InterfaceInfo Class on page 166.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

GetAvailableInterfacesCount Method

Syntax

```
from rttlib import dsethernetapilib

MyInterfaceCount = dsethernetapilib.GetAvailableInterfacesCount()
```

Purpose

To get the number of available Ethernet interfaces.

Parameter

–

Return value

The function returns a value of the following type:

Type	Description
Integer	The number of available Ethernet interfaces.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

GetInterfaceCapabilities Method

Syntax

```
from rttlib import dsethernetapilib

MyInterfaceCapabilities =
dsethernetapilib.GetInterfaceCapabilities(InterfaceHandle)
```

Purpose

To get the capabilities of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to retrieve the capabilities from.

Return value

The function returns a value of the following type:

Type	Description
Integer	Flags for interface capabilities: ▪ `0x01`: `dsethernetapilib.icHOST`: The interface is a host PC network adapter. ▪ `0x02`: `dsethernetapilib.icLOOPBACK`: The interface is a loopback network adapter. ▪ `0x04`: `dsethernetapilib.icCTRLTIMESTAMPS`: The interface provides network controller timestamps. ▪ `0x08`: `dsethernetapilib.icFRAMESFILTERING`: The interface supports Ethernet frames filtering. ▪ `0x10`: `dsethernetapilib.icSOURCEMAC`: The interface supports automatic assignment of source MAC address.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

GetInterfaceMacAddress Method

Syntax

```
from rttlib import dsethernetapilib
MyInterfaceMacAddress =
dsethernetapilib.GetInterfaceMacAddress(InterfaceHandle)
```

Purpose

To get the MAC address of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to get the MAC address from.

Return value

The function returns a value of the following type:

Type	Description
List	List of the 6 bytes representing the MAC address of the interface.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetSupportedFrameTypes Method

Syntax

```
from rttlib import dsethernetapilib

MySupportedFrameTypes =
dsethernetapilib.GetSupportedFrameTypes(InterfaceHandle)
```

Purpose

To get the supported frame types of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to get the supported frame types from.

Return value

The function returns a value of the following type:

Type	Description
List	List of supported frames types. The following types are defined: ▪ 1: `dsethernetapilib.ftETHERNET`: Ethernet frame with frame check sequence ▪ 2: `dsethernetapilib.ftLOOPBACK`: Loopback frame (currently not supported) ▪ 3: `dsethernetapilib.ftETHERNETNOFCS` Ethernet frame without frame check sequence (currently not supported)

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetSupportedFrameTypesCount Method

Syntax

```
from rttlib import dsethernetapilib

MySupportedFrameTypesCount =
dsethernetapilib.GetSupportedFrameTypesCount(InterfaceHandle)
```

Purpose

To get the number of supported frame types of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to get the number of supported frame types from.

Return value

The function returns a value of the following type:

Type	Description
Integer	The number of supported frame types.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetTime Method

Syntax

```
from rttlib import dsethernetapilib

MyTime = dsethernetapilib.GetTime(InterfaceHandle)
```

Purpose

To get the hardware time of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to get the hardware time from.

Return value

The function returns a value of the following type:

Type	Description
Tuple	The hardware time in a tuple that consists of: ▪ Time (Integer) ▪ Controller time (Integer)

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

GetTimeResolution Method

Syntax

```
from rttlib import dsethernetapilib

MyTimeResolution =
dsethernetapilib.GetTimeResolution(InterfaceHandle)
```

Purpose

To get the hardware time resolution of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to get the hardware time resolution from.

Return value

The function returns a value of the following type:

Type	Description
Tuple	The hardware time resolution in a tuple that consists of: ▪ Time resolution (Integer) ▪ Controller time resolution (Integer)

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

InitInterface Method

Syntax

```
from rttlib import dsethernetapilib
IsInitDone = dsethernetapilib.InitInterface(InterfaceHandle)
```

Purpose

To initialize an Ethernet interface.

Description

The initialization of an Ethernet interface may last several model steps. You must therefore call `dsethernetapilib.InitInterface()` in every model step until the return value is True. As long as the initialization is not completed, the method returns None.

A timeout for the repetitive call to `dsethernetapilib.InitInterface()` may prevent an endless loop or a stalling RTT sequence if the initialization does not succeed and does not throw an Exception.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to be initialized.

Return value

The function returns a value of the following type:

Type	Description
Boolean	The state of the initialization: ▪ None: The initialization is not completed yet. ▪ True: The initialization is successfully completed.

Example

The following example shows a function that use the method to initialize Ethernet interfaces.

```
def initEthernet(TimeOutSteps = 60):
    """
    Function: initEthernet
        This function shows how to initialize the Ethernet interfaces with
        the rttlib.dsethernetapilib.
    """
    # Use global ethernetlib objects.
    global InterfaceInfoObjects
    global InterfaceHandles
    # Retrieve the information about all available Ethernet interfaces.
    InterfaceInfoObjects = dsethernetapilib.GetAvailableInterfaces()
    yield None
    # Initialize all available Ethernet interfaces.
    for interface in InterfaceInfoObjects:
        InterfaceHandle = dsethernetapilib.RegisterInterface(interface.AccessProviderName, \
                                         interface.InterfaceName, \
                                         interface.InterfaceSerialNumber, \
                                         interface.InterfaceIdentifier)
        # Add an interface handle to the global list.
        InterfaceHandles.append(InterfaceHandle)
        # The initialization of an Ethernet interface might last several model steps.
        # When the function returns 'True' the initialization was successful.
        # As long as the return value of InitInterface() is 'None' the interface is
        # not initialized. A timeout is useful to avoid waiting too long for
        # a interface initialization that might not be possible.
        IsInitialized = None
        while(IsInitialized == None):
            IsInitialized = dsethernetapilib.InitInterface(InterfaceHandle)
            TimeOutSteps -= 1
            if(TimeOutSteps <= 0):
                raise Exception("Could not initialize Ethernet interfaces.")
            yield None
        yield None
        # Activate valid ethernet interface.
        dsethernetapilib.ActivateInterface(InterfaceHandle)
        yield None
```

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

IsInterfaceAccessible Method

Syntax

```
from rttlib import dsethernetapilib

IsAccessible =
dsethernetapilib.IsInterfaceAccessible(InterfaceHandle)
```

Purpose

To check whether an Ethernet interface is accessible.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of the Ethernet interface to be checked.

Return value

The function returns a value of the following type:

Type	Description
Boolean	State of the interface: ▪ `True`: The interface is accessible. ▪ `False`: The interface is not accessible.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

IsInterfaceAvailable Method

Syntax

```
from rttlib import dsethernetapilib

IsAvailable = dsethernetapilib.IsInterfaceAvailable(InterfaceHandle)
```

Purpose

To check whether an Ethernet interface is available.

Parameter

The function uses the following parameter:

Parameter	Type	Description
AccessProviderName	String	The name of the access provider.
InterfaceName	String	The name of the Ethernet interface.
InterfaceSerialNumber	String	The serial number of the Ethernet interface.
InterfaceIdentifier	String	The identifier of the Ethernet interface.

Return value

The function returns a value of the following type:

Type	Description
Boolean	The availability of the interface: ▪ `True`: The interface is available. ▪ `False`: The interface is not available.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

ReadFrames Method

Syntax

```
from rttlib import dsethernetapilib

MyRxFrames = dsethernetapilib.ReadFrames(InterfaceHandle, RxBuffer)
```

Purpose

To read the Ethernet frames from the receive queue of an Ethernet interface.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of Ethernet interface to read from.
RxBuffer	Bytes	Buffer object to hold the received frames from the receive queue.

Description

This function reads the received Ethernet frames from the receive queue of the specified Ethernet interface and returns them to the list `MyRxFrames`.

Return value

The function returns a value of the following type:

Type	Description
List	List of EthRawFrame[1] objects representing all the Ethernet frames that could be read from the receive queue and using the RxBuffer.

[1] Refer to EthRawFrame Class on page 168

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

RegisterInterface Method

Syntax

```
from rttlib import dsethernetapilib

MyInterfaceHandle =
dsethernetapilib.RegisterInterface(AccessProviderName, \
                                         InterfaceName, \
                                         InterfaceSerialNumber, \
                                         InterfaceIdentifier)
```

Purpose

To register an Ethernet interface using its interface information.

Description

Before you can use an Ethernet interface, you must register it. To get the serial numbers, user strings and identifier of your Ethernet interfaces, use the `GetAvailableInterfaces` method.

Whenever you register an Ethernet interface, you must also unregister it later on via the `UnregisterInterface` method.

Parameter

The function uses the following parameter:

Parameter	Type	Description
AccessProviderName	String	The access provider name of the Ethernet interface.
InterfaceName	String	The interface name of the Ethernet interface.

Parameter	Type	Description
InterfaceSerialNumber	String	The interface serial number of the Ethernet interface.
InterfaceIdentifier	String	The interface identifier of the Ethernet interface.

Return value

The function returns a value of the following type:

Type	Description
Integer	The handle of the registered Ethernet interface.

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

References

GetAvailableInterfaces Method.. 173
UnregisterInterface Method.. 186

SetFilter Method

Syntax

```
from rttlib import dsethernetapilib

dsethernetapilib.SetFilter(InterfaceHandle, Filter)
```

Purpose

To set a frame filter for an Ethernet interface.

Description

You can filter the frames of an Ethernet interface using the BPF (Berkeley Packet Filter) syntax.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	The handle of the Ethernet interface to which the filter shall be applied.
Filter	String	The filter string in BPF syntax that defines the frames to be filtered.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 🕮)

TransmitFrames Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.TransmitFrames(InterfaceHandle, TxBuffer, Frames)
```

Purpose

To transmit the Ethernet frames using an Ethernet interface and a transmit buffer.

Description

This method transmits the given Ethernet frames (`Frames`) using the specified Ethernet interface. The `TxBuffer` must be created using the CreateBuffer method. The buffer must be large enough to hold all frames to be transmitted.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	The handle of the Ethernet interface to which the filter shall be applied.
TxBuffer	Bytes	The buffer object that holds the frames to be transmitted.
Frames	List	The list of EthRawFrame[1] objects to be transmitted.

[1] Refer to EthRawFrame Class on page 168.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

References

CreateBuffer Method.. 170

UnregisterInterface Method

Syntax

```
from rttlib import dsethernetapilib
dsethernetapilib.UnregisterInterface(InterfaceHandle)
```

Purpose

To unregister an Ethernet interface.

Description

This method frees all the dependencies of the selected Ethernet interface. The handle becomes invalid.

You can unregister an Ethernet interface only if it was previously registered via the RegisterInterface method.

Parameter

The function uses the following parameter:

Parameter	Type	Description
InterfaceHandle	Integer	Handle of Ethernet interface to unregister.

Return value

–

Related topics

Basics

Basics on the dsethernetapilib Module (Real-Time Testing Guide 📖)

References

RegisterInterface Method.. 183

rttlib.globalvariables Module

rttlib.globalvariables Module Description

Introduction

Enable global variables between different RTT sequences via the rttlib.globalvariables module.

Description

This module can be used to share global data between isolated RTT sequences. The rttlib.globalvariables module is a dSPACE-provided module that is shared between all RTT sequences. It persists as long as the Python interpreter is running.

Creating global variables

You can create global variables by inserting a new attribute into the namespace of the `rttlib.globalvariables` module, for example:

```
from rttlib import globalvariables
# Create "myvariable1"
globalvariables.myvariable1 = 42.0
```

Accessing global variables

Global variables can be accessed like normal module attributes, for example:

```
from rttlib import globalvariables
print(globalvariables.myvariable1)
```

Related topics

Basics

Using Variables Accessible by Several RTT Sequences (Real-Time Testing Guide 🕮)

rttlib.hostcall Module

Introduction

This module provides functions for sending data to a registered Python script on the host PC.

Hostcall Function

Syntax

```
from rttlib import hostcall
yield hostcall.Hostcall(ReturnResult,\
    [hostcall_arg_1, hostcall_arg_2, …, hostcall_arg_n])
```

Purpose

To send a Python data object to the host PC and wait for the return value from the host PC.

Description

The RTT sequence pauses its execution until the host PC has sent the return value.

The send and return values must be restorable with a cPickle module.

Note

If the call of `hostcall.Hostcall()` is part of the scheduler.ParallelRace() generator object, the host call can be aborted while it is still in process. An aborted host call can block other pending host calls, so do not use host calls in ParallelRace() constructs.

Parameter

The function uses the following parameters:

Parameter	Type	Description
ReturnResult	List	List to store the results returned from the host
[hostcall_arg_1, hostcall_arg_2, …, hostcall_arg_n]	Any	An arbitrary number of arguments. These arguments are passed to the corresponding OnHostCall handler as a tuple.

Note

It must be possible to serialize the arguments [hostcall_arg_1, hostcall_arg_2, …, hostcall_arg_n] with a cPickle module.

Return value

–

Exceptions

The function can raise the following exception:

Exception	Description
exceptions.RuntimeError	The host PC cannot receive the Python data object.

Related topics

References

ParallelRace Generator Function.. 199

rttlib.rs232lib Module

Introduction

This module provides functions for sending and receiving data via an RS232 interface of the real-time platform.

Where to go from here

Information in this section

OpenEx Function.......190
To open the serial port and create a handle object.

SetConfig Function.......191
To configure the serial channel.

GetNumInBytes Function.......192
To get the number of bytes in the read buffer.

Read Function.......193
To read data from the specified serial channel.

Write Function.......194
To transmit one byte through the specified serial channel.

WriteString Function.......195
To transmit a string through the specified serial channel.

PurgeComm Function.......196
To clear all the buffers of the specified serial channel.

Close Function.......196
To close the open connection of an RS232 interface.

OpenEx Function

Syntax

```
from rttlib import rs232lib
hComRS232 = rs232lib.OpenEx(board_type, board_index, port_index)
```

Purpose

To open the serial port and create a handle object.

Description

This function must be used before you can configure the serial channel. It must be called in the initialization section of an RTT sequence (not within the `MainGenerator` function).

If the specified serial port is used by another RTT sequence, an exception is thrown and the RTT sequence cannot be created.

Parameters

The function uses the following parameters:

Parameter	Type	Description
board_type	Integer	Type of board to be addressed. To specify the type, use the following constant defined in the rs232lib module: ▪ ONBOARD: To use the serial interface which is located on a processor board (DS1006).
board_index	Integer	Number of the board of the same type in a modular system which is used. For a DS1006, the value is always 1.
port_index	Integer	Number of the controller on the board which is used For a DS1006, the value is always 1.

Return value

The function returns a value of the following type:

Type	Description
Python object	A handle object which contains channel information and ownership. This handle is to be used for each subsequent configuration, read/write and info function call.

Related topics

References

Close Function..196
SetConfig Function..191

SetConfig Function

Syntax

```
from rttlib import rs232lib
rs232lib.SetConfig(hComRS232, baud_rate, word_length, parity,
stop_bits)
```

Purpose

To configure the serial channel.

Description

The function configures the specified channel with the given parameters. If the `word_length`, `parity` and `stop_bits` parameters are not specified, the function sets the default to 8N1 mode (8 bit words, no parity, 1 stop bit).

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Python object	Handle object which contains the channel object to be used. It is the return value of the `OpenEx` function.
baud_rate	Integer	Baud rate at which the communication port operates. Valid values are multiple of 300 in the range 300 ... 115200, such as 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200.
word_length	Integer	Number of data bits to be used. Valid values are: 5, 6, 7, 8[1].
parity	String	Parity scheme to be used. Valid values are specified by constants: ▪ `NO_PARITY`: No parity ▪ `ODD`: Parity bit is set so that there is an odd number of `"1"` bits in the byte, including the parity bit. ▪ `EVEN`: Parity bit is set so that there is an even number of `"1"` bits in the byte, including the parity bit. ▪ `MARK`: Parity bit forced to 1 ▪ `SPACE`: Parity bit forced to 0
stop_bits	Float	Number of stop bits to be used. Valid values are: 1, 1.5, 2[1]

[1] The use of 5 data bits with 2 stop bits is an invalid combination, as are 6, 7, 8 data bits with 1.5 stop bits.

Return value

–

Related topics

References

OpenEx Function.. 190

GetNumInBytes Function

Syntax

```
from rttlib import rs232lib
RetVal = rs232lib.GetNumInBytes(hComRS232)
```

Purpose

To get the number of bytes in the read buffer.

Parameter

The function uses the following parameter:

Parameter	Type	Description
hComRS232	Python object	Handle object which contains the channel object to be used. It is the return value of the `OpenEx` function.

Return value

The function returns a value of the following type:

Type	Description
Integer	Number of bytes in read buffer

Related topics

References

OpenEx Function 190
Read Function 193

Read Function

Syntax

```
from rttlib import rs232lib
RetVal = rs232lib.Read(hComRS232, requested_bytes)
```

Purpose

To read data from the specified serial channel.

Description

If the receive buffer contains less data than requested, the function returns immediately with the complete buffer content and does not wait for further data.

Parameter

The function uses the following parameter:

Parameter	Type	Description
hComRS232	Python object	Handle object which contains the channel object to be used. It is the return value of the `OpenEx` function.

Parameter	Type	Description
requested_bytes	Int	Number of bytes which are read from the buffer (up to 63 bytes).

Return value

The function returns a value of the following type:

Type	Description
Python string object	An object containing the buffer content of the serial channel. The object is 'None' if the buffer is empty or nothing was requested.

Related topics

References

OpenEx Function.. 190

Write Function

Syntax

```
from rttlib import rs232lib
rs232lib.Write(hComRS232, tx_byte)
```

Purpose

To transmit one byte through the specified serial channel.

Description

If your selected transmission word length is smaller than 8 bits, obsolete high bits will be cut off.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Python object	Handle object containing the channel object to be used. It is the return value of the `OpenEx` function.
tx_byte	Integer	Byte to be transmitted.

Return value

-

Related topics

References

OpenEx Function.......... 190

WriteString Function

Syntax

```
from rttlib import rs232lib
rs232lib.WriteString(hComRS232, StringToWrite)
```

Purpose

To transmit a string through the specified serial channel.

Description

The function sends a string over the specified serial channel. If your selected transmission word length is smaller than 8 bits, obsolete high bits will be cut off per character.

Note

There is no guarantee that these values are immediately transferred to the connected communication partner.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Python object	Handle object containing the channel object to be used. It is the return value of the `OpenEx` function.
StringToWrite	String	String to be transmitted (up to 63 bytes).

Return value

-

Related topics

References

OpenEx Function.......... 190

PurgeComm Function

Syntax

```
from rttlib import rs232lib

rs232lib.PurgeComm(hComRS232)
```

Purpose

To clear the buffers of the specified serial channel.

Description

The function clears all the receive and transmit buffer of a serial channel.

Parameter

The function uses the following parameter:

Parameter	Type	Description
hComRS232	Python object	Handle object containing the channel object to be used. It is the return value of the `OpenEx` function.

Return value

–

Related topics

References

OpenEx Function... 190

Close Function

Syntax

```
from rttlib import rs232lib

rs232lib.Close(hComRS232)
```

Purpose

To close the open connection of an RS232 interface.

Descriptions

If the function is successful, the handle object is rendered invalid. All the functions which try to use the handle afterwards will fail and raise an exception. The remaining bytes in the buffer are discarded.

If the function is not successful, the handle object is not invalidated.

If the `Close` method is not called for a channel, the channel is locked and cannot be used by other RTT sequences. The method is therefore called automatically when the RTT sequence is removed from the simulation platform.

Parameter

The function uses the following parameter:

Parameter	Type	Description
hComRS232	Python object	Handle object which contains the channel object to be used. It is the return value of the `OpenEx` function.

Return value

–

Related topics

References

OpenEx Function.. 190

rttlib.scheduler Module

Introduction

This module provides generator functions to execute concurrent operations in an RTT sequence.

Where to go from here

Information in this section

Parallel Generator Function.. 198
To create a Parallel() generator object from the specified arguments.

ParallelRace Generator Function.. 199
To create a ParallelRace() generator object from the specified arguments.

Parallel Generator Function

Syntax

```
from rttlib import scheduler
yield scheduler.Parallel(*generator_objects)
```

Purpose

To create a Parallel() generator object from the specified arguments.

Description

Parallel() takes an arbitrary number of arguments and executes them concurrently. The number of arguments is variable. Every generator object is executed exactly once for each step of the base rate. This concurrent execution is performed until *all* generator objects are finished.

The execution order in a sampling step is exactly the order in which the arguments were passed to Parallel().

Parameter

The function uses the following parameter:

Parameter	Type	Description
generator_object1, generator_object2, ... generator_object*n*	Generator object	This function takes an arbitrary number of arguments. Each argument has to be a generator function.

Return value

–

Exceptions

–

Example

```
from rttlib import scheduler
from rttlib import variable
from rttlib import utilities
#--------------------------------------------------------
# Module global variables
#--------------------------------------------------------
CurrentTime = utilities.currentTime
WarningLightSwitch = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
TurnSignalLeft     = variable.Variable(r'Model Root/FrontLightEcu/TurnSignalLeft')
BatteryVoltage     = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
#
# Ramp generator
# - stimulates a ramp onto 'signal'
# - for 'duration' seconds
#
def GenerateRamp(signal, duration):
    start_time = CurrentTime.Value
    while CurrentTime.Value < (start_time + duration):
        signal.Value += 0.01
        yield None
def MainGenerator(*args):
    # Set initial signal values
    WarningLightSwitch.Value = 0.0
    TurnSignalLeft.Value = 0.0
    BatteryVoltage.Value = 0.0
    print(" CurrentTime : ", CurrentTime.Value)
    # Generate ramps simultaneously onto three signals
    yield scheduler.Parallel(GenerateRamp(WarningLightSwitch, 2.0), \
                                GenerateRamp(TurnSignalLeft, 4.0), \
                                GenerateRamp(BatteryVoltage, 6.0))
    print(" CurrentTime : ", CurrentTime.Value)
```

Related topics

Basics

Using the Parallel() Generator Function (Real-Time Testing Guide 🕮)

ParallelRace Generator Function

Syntax

```
from rttlib import scheduler
yield scheduler.ParallelRace(*generator_objects)
```

Purpose

To create a ParallelRace() generator object from the specified arguments.

Description

ParallelRace() takes an arbitrary number of arguments and executes them concurrently. The number of generators is variable. Every generator object is executed exactly once for each step of the base rate. This concurrent execution is performed until the *first* generator object is finished.

The execution order in a sampling step is exactly the order in which the arguments were passed to ParallelRace().

> **Note**
>
> Do not use host calls in ParallelRace constructs. If the call of `hostcall.Hostcall()` is part of the scheduler.ParallelRace() generator object, the host call can be aborted while it is still in process. An aborted host call can block other pending host calls.

Parameter

The function uses the following parameter:

Parameter	Type	Description
generator_object1, generator_object2, ... generator_object*n*	Generator object	This function takes an arbitrary number of arguments. Each argument has to be a generator function.

Return value

–

Exceptions

–

Example

```
from rttlib import scheduler
from rttlib import variable
from rttlib import utilities
#--------------------------------------------------------
# Module global variables
#--------------------------------------------------------
CurrentTime = utilities.currentTime
WarningLightSwitch = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
TurnSignalLeft     = variable.Variable(r'Model Root/FrontLightEcu/TurnSignalLeft')
BatteryVoltage     = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
#
# Ramp generator
# - stimulates a ramp onto 'signal'
# - for 'duration' seconds
#
def GenerateRamp(signal, duration):
    start_time = CurrentTime.Value
    while CurrentTime.Value < (start_time + duration):
        signal.Value += 0.01
        yield None
```

```
def MainGenerator(*args):
    # Set initial signal values
    WarningLightSwitch.Value = 0.0
    TurnSignalLeft.Value = 0.0
    BatteryVoltage.Value = 0.0
    print(" CurrentTime : ", CurrentTime.Value)
    # Generate ramps simultaneously onto three signals
    yield scheduler.ParallelRace(GenerateRamp(WarningLightSwitch, 2.0), \
                    GenerateRamp(TurnSignalLeft, 4.0), \
                    GenerateRamp(BatteryVoltage, 6.0))
    print(" CurrentTime : ", CurrentTime.Value)
```

Related topics

Basics

Using the ParallelRace() Generator Function (Real-Time Testing Guide 📖)

rttlib.utilities Module

Introduction

This module provides functions for common real-time operations.

Where to go from here

Information in this section

GetSequenceArgument Function..202
To return the (optional) RTT sequence argument that was passed to the Real-Time Test Manager Server's Create function

currentTime Variable Object..203
To return a variable object similar to a variable object from the currentTime variable of a Simulink model.

Logging..204
To print a message to the standard output or to the dSPACE Log file.

modelStepSize Variable Object...206
To return a variable object similar to a variable object from the modelStepSize variable of a Simulink model.

SequenceProperties..206
To get/set properties of the RTT sequence.

SetCallGCAfterRemovingAllSequences Function..................................208
To call the Garbage Collector to free memory when the last RTT sequence is removed.

SetImportTimeslice Function...209
To increase the computation time per sampling step reserved for the initialization of an RTT sequence.

Wait Function...211
To suspend the RTT sequence for a specified number of seconds.

GetSequenceArgument Function

Syntax

```
from rttlib import utilities

argument = utilities.GetSequenceArgument()
```

Purpose

To return the (optional) RTT sequence argument that was passed to the Real-Time Test Manager Server's `Create` function.

Description

The execution time of `GetSequenceArgument()` depends on the size of the argument. This function should therefore be called during the initialization phase.

Parameters

–

Return value

The function returns a value of the following type:

Type	Description
Tuple	The Python object that was passed by the Real-Time Test Manager Server's `Create` function.

Exception

The function can raise the following exception:

Type	Description
exceptions.ValueError	No parameters were passed.

Related topics

Basics

Creating and Starting RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

References

Create Method.. 57

currentTime Variable Object

Syntax

```
from rttlib import utilities

CurrentTime = utilities.currentTime
```

Purpose

To return a variable object similar to a variable object from the currentTime variable of a Simulink model.

Description

The object has to be called during the initialization phase of a module. Using the currentTime variable object is faster than creating the variable object via the Simulink variable path.

Parameters

–

Return value

The object is of the following type:

Type	Description
Variable object	A currentTime variable object based on Variable (read-only).

Related topics

References

Variable Class..212

Logging

Syntax

```
from rttlib import utilities
utilities.Logging.Enable = True
utilities.Logging.Direction = utilities.Logging.TO_LOGFILE |
utilities.Logging.TO_ON_WRITE
utilities.Logging.info(*objects, sep=' ', end='\n')
utilities.Logging.warning(*objects, sep=' ', end='\n')
utilities.Logging.error(*objects, sep=' ', end='\n')
```

Purpose

To print a message to the standard output or to the dSPACE Log file.

Description

You can print the message either to the standard output or to the dSPACE Log file.

All non-keyword arguments are converted to strings like the Python method `str()` does, separated by `sep` and followed by `end`. Both `sep` and `end` must be strings. They can also be None, which means to use the default values.

Attributes

The Logging class provides the following attributes:

Attribute	Description

Enable	Enables or disables printing.
	The following code shows some examples:
	`# To enable printing`
	`utilities.Logging.Enable = True`
	`# To disable printing`
	`utilities.Logging.Enable = False`

Attribute	Description

Direction	Specifies the target for the message to be printed (standard output or dSPACE log file).
	The following code shows some examples:
	`# To use the standard output as target`
	`utilities.Logging.Direction = utilities.Logging.TO_ON_WRITE`
	`# To use the log file as target`
	`utilities.Logging.Direction = utilities.Logging.TO_LOGFILE`
	`# To use the standard output and log files as targets`
	`utilities.Logging.Direction = \`
	`    utilities.Logging.TO_LOGFILE ｜ utilities.Logging.TO_ON_WRITE`

Methods

The `Logging` class has the following methods. The method used specifies the severity of the message. All the methods have the same parameters, see below.

Method	Purpose
`info(*objects, sep=' ', end='\n')`	To print an info message.
`warning(*objects, sep=' ', end='\n')`	To print a warning message.
`error(*objects, sep=' ', end='\n')`	To print an error message.

Parameters

The methods use the following parameters:

Parameter	Type	Description
objects	object	Specifies the object that contains the message to be printed. The maximum length of the message to be written is 4088 characters.
sep	String	Specifies a string that is used to separate elements of the object.
end	String	Specifies a string that is used to end the message to be printed.

Related topics

Basics

Printing Messages in the dSPACE Log from an RTT Sequence (Real-Time Testing Guide 🕮)

modelStepSize Variable Object

Syntax

```
from rttlib import utilities

StepSize = utilities.modelStepSize
```

Purpose

To return a variable object similar to a variable object from the modelStepSize variable of a Simulink model.

Description

The object has to be called during the initialization phase of a module. Using the modelStepSize variable object is faster than creating the variable object via the Simulink variable path.

Parameters

–

Return value

The object is of the following type:

Type	Description
Variable object[1]	A modelStepSize variable object based on Variable (read-only).

[1] Refer to Variable Class on page 212.

Related topics

References

modelStepSize (RTI and RTI-MP Implementation Reference 📖)
Variable Class..212

SequenceProperties

Syntax

```
from rttlib import utilities
SequenceProperties = utilities.SequenceProperties()
```

Purpose

To get/set properties of the RTT sequence.

Description

You can use a `SequenceProperties` object to *get* the following information on the RTT sequence: name, description, file name, and priority.

You can use a `SequenceProperties` object to *get* and *set* the sequence channel of the RTT sequence.

Real-Time Testing version

This method is supported as of Real-Time Testing 2.5.

Parameters

–

Attributes

The following attributes are part of the class:

Attribute	Type	Description
Name	String	To get the name of the RTT sequence.
Description	String	To get the description of the RTT sequence.
FileName	String	To get the file name of the RTT sequence.
Priority	Integer	To get the priority of the RTT sequence.
SequenceChannel	Integer	To get the sequence channel (time when the RTT sequence is executed): ▪ `utilities.constants.scPreComputation`: The RTT sequence is executed before the simulation model is calculated by the real-time application. ▪ `utilities.constants.scPostComputation`: The RTT sequence is executed after the simulation model is calculated by the real-time application.

Example

The following example shows how to use the class.

```
from rttlib import utilities
SequenceProperties = utilities.SequenceProperties()
# To get the name of the RTT sequence.
SequenceName = SequenceProperties.Name
# To get the description
SequenceDescription = SequenceProperties.Description
# To get the file name
SequenceFileName = SequenceProperties.FileName
# To get the priority of the RTT sequence
SequencePriority = SequenceProperties.Priority
# To get the sequence channel
SequenceChannel = SequenceProperties.SequenceChannel
# To set the sequence channel for executing the RTT sequence before the model.
SequenceProperties.SequenceChannel = utilities.constants.scPreComputation
# To set the sequence channel for executing the RTT sequence after the model.
SequenceProperties.SequenceChannel = utilities.constants.scPostComputation
```

Related topics

Basics

Creating and Starting RTT Sequences in Python Scripts (Real-Time Testing Guide 📖)

SetCallGCAfterRemovingAllSequences Function

Syntax

```
from rttlib import utilities
utilities.SetCallGCAfterRemovingAllSequences(True)
```

Purpose

To call the Garbage Collector to free memory when the last RTT sequence is removed.

Description

Generally, you have to avoid circular references by the objects allocated within an RTT sequence. The memory allocated for such objects cannot automatically be freed after removing the RTT sequence. As a consequence, this can lead to a memory leak problem.

If you do need to use circular references, you can use the following function in your RTT sequence. It forces the system to free these objects:

```
utilities.SetCallGCAfterRemovingAllSequences(True)
```

When all RTT sequences are removed, the Garbage Collector is called and it frees the objects, even if they have circular references. As long as this flag is set, the Garbage Collector is called every time after you remove the last existing RTT sequence. This call is time-consuming and can lead to undesired test overruns. You should set it only if you suspect that some objects of the RTT sequences are not completely removed due to possible circular references or other issues.

The call of the Garbage Collector, you must clear the flag by using another RTT sequence:

```
utilities.SetCallGCAfterRemovingAllSequences(False)
```

After that, the Garbage Collector is not called again when all the RTT sequences are removed.

Parameters

The method uses the following parameter:

Parameter	Type	Description
Enable	Boolean	Enables or disables the call of the Garbage Collector ▪ `True`: Calling is enabled. ▪ `False`: Calling is disabled.

Return value

–

Example

The following example shows an RTT sequence with a circular reference. To free the memory, calling the Garbage Collector is enabled.

```
from rttlib import utilities
list_with_circular_ref = []
for i in range(100):
    list_with_circular_ref.append("0123456789"*10)
# A circular reference in the object.
list_with_circular_ref.append(list_with_circular_ref)
# Enable calling of the Garbage Collector after removing all RTT sequences
# to remove the circular reference of the previous object.
# In another RTT sequence, the call of the Garbage Collector must be disabled.
utilities.SetCallGCAfterRemovingAllSequences(True)
def MainGenerator():
...
    yield None
```

When the previous RTT sequence is removed from the platform, the call of the Garbage Collector can be disabled. This is shown in the following RTT sequence.

```
from rttlib import utilities
# Disable calling the Garbage Collector after all sequences are removed.
utilities.SetCallGCAfterRemovingAllSequences(False)
def MainGenerator():
...
yield None
```

Related topics

References

OnHostCall Method.. 76

SetImportTimeslice Function

Syntax

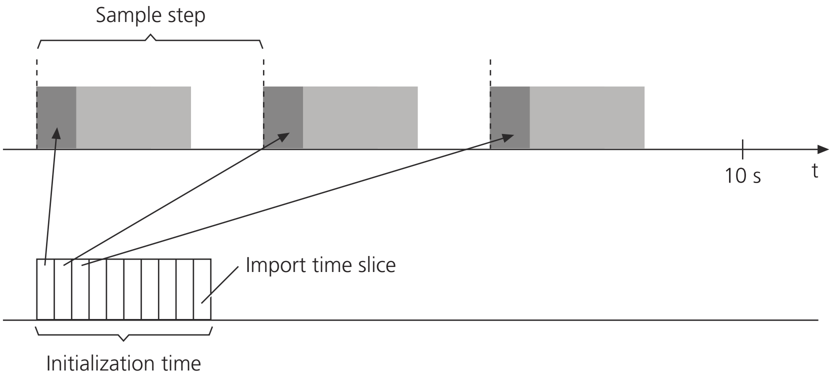
```
from rttlib import utilities
utilities.SetImportTimeslice(seconds)
```

Purpose

To increase the computation time per sampling step reserved for the initialization of an RTT sequence.

Description

The initialization of an RTT sequence is performed in import time slices. An import time slice is a reserved fraction of computation time in a sampling step. The time slice in each model step for the import phase is 20 µs by default. If an RTT sequence must be initialized in a fixed time period and the default value of the import time slice is not sufficient, you can increase the value. The theoretical maximum value for the import time slice is the modelStepSize, but the computation time needed for the model itself must be subtracted.



The modified import time slice is only valid for the initialization phase of the current RTT sequence. Subsequent RTT sequences start with the default import time slice of 20 µs. Setting the length of the import time slices can be necessary, for example, to increase the number of variable objects that can be created during the initialization phase. For further information on the relation between initialization time, import time slice, sampling step, and timeout, refer to Avoiding a Timeout During Initialization of an RTT Sequence (Real-Time Testing Guide 🕮).

Parameters

The method uses the following parameter:

Parameter	Type	Description
seconds	Float	Import time slice in seconds

Return value

–

Example

```
import math
from rttlib import utilities
# modify import time slice
utilities.SetImportTimeslice(0.0002) # 200 µs
# this loop takes longer than 10 seconds with the
# standard time slice of 20 µs -> timeout
for i in range(400000):
    dummy = 42.0 * 43.0 * math.sqrt(17)
```

Related topics

References

modelStepSize Variable Object.......206
OnHostCall Method.......76

Wait Function

Syntax

```
from rttlib import utilities
yield utilities.Wait(Duration)
```

Purpose

To suspend the RTT sequence for a specified number of seconds.

Description

The `Wait()` function is a generator function which suspends an RTT sequence until the specified number of seconds have passed. It is a convenience function that you can use instead of a handcoded while loop (refer to the example below). The major benefits compared to a while loop are:

- The RTT sequence is simpler.
- It is faster than the implementation in Python.

The time base used for time measurement is obtained from the `currentTime` variable object.

Parameters

The method uses the following parameter:

Parameter	Type	Description
Duration	Float	Wait time in seconds

Return value

-

Example

```
from rttlib import utilities
# Simple while loop that waits 10 seconds
StartTime = CurrentTime.Value
while ((StartTime + 10.0) > CurrentTime.Value):
    yield None
# The same functionality with a call to Wait()
yield utilities.Wait(10.0) # 10 seconds
```

Related topics

References

currentTime Variable Object..203

rttlib.variable Module

Introduction

This module provides a class to represent a variable object for accessing Simulink variables from an RTT sequence.

Where to go from here

Information in this section

Variable Class..212
To represent a variable object to access Simulink variables from an RTT sequence.

IsA2L Method..214
To return whether the variable description file for the real-time application is a TRC or A2L file.

Name Method..215
To return the name of the simulator variable.

Variable Class

Syntax

```
from rttlib import variable
Var = variable.Variable(VariableName, disableScaling = False)
```

Purpose

To represent a variable object to access Simulink variables from an RTT sequence.

Description

Some limitations apply when accessing Simulink variables, refer to General Limitations for Real-Time Testing (Real-Time Testing Guide 🕮).

Parameter

The class uses the following parameters:

Parameter	Type	Description
VariableName	String	Name of the simulator variable including subsystems specified within the system description file If you want to access a variable of a remote CPU in a multiprocessor system, VariableName must also contain the application name (name of the submodel) running on the CPU. For an example, refer to Example 2 on page 214.

Parameter	Type	Description
disableScaling	Boolean	Specifies whether a scaling that is defined for the variable is considered (optional). The default is False. ▪ `False`: The scaling is considered. ▪ `True`: The scaling is not considered. For details on the scaling of variables, refer to Scaling (Real-Time Testing Guide 📖).

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Value	Float or Integer	To get and set the variable value. The value can be a floating or integer value, regardless of the type of the Simulink variable. If the Simulink variable type is integer and the value written to the variable is of floating type, the floating point value be rounded to fit the Simulink integer variable. If the value exceeds the data range of the Simulink variable, an exception is raised. The return value is always a float type regardless of the type of the Simulink variable.

Methods

The following method is part of the class:

Method	Purpose
Name	To return the name of the simulator variable. Refer to Name Method on page 215.
IsA2L	To return whether the variable description file for the real-time application is a TRC or A2L file. Refer to IsA2L Method on page 214.

Examples

Example 1 The following example shows how to use access variables of an A2L file and TRC file.

```
from rttlib import variable
if variable.IsA2L():
    # The variable description for this application is an A2L file.
    WarningLightSwitch = variable.Variable(r'WarningLightSwitchValue')
    TurnSignalLeft     = variable.Variable(r'RearLightEcuTurnSignalLeft')
    BatteryVoltage     = variable.Variable(r'BatteryVoltageValue')
    TurnSignalLever    = variable.Variable(r'TurnSignalLeverValue')
else:
    # The variable description for this application is a TRC file.
    WarningLightSwitch = variable.Variable(r'Model Root/WarningLightSwitch[0|1]/Value')
    TurnSignalLeft     = variable.Variable(r'Model Root/RearLightEcu/TurnSignalLeft')
    BatteryVoltage     = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
    TurnSignalLever    = variable.Variable(r'Model Root/TurnSignalLever[-1..1]/Value')
```

Example 2 The following example shows how to access a variable of a remote CPU in a multiprocessor system. `MyAppl1` and `MyAppl2` are the names of the applications, and `SimulationTime` is the variable name.

```
from rttlib import variable

if variable.IsA2L():
    # The variable description for this application is an A2L file.
    # Create variable object on node 'MyAppl1'.
    currentTimeMaster = variable.Variable(r'MyAppl1()://masterAppl/SimulationTime')
    # Create variable object on node 'MyAppl2'.
    currentTimeSlave = variable.Variable(r'MyAppl2()://masterAppl/SimulationTime')
else:
    # The variable description for this application is a TRC file.
    # Create variable object on node 'MyAppl1'.
    currentTimeMaster = variable.Variable(r'MyAppl1/Model Root/master/Clock/Out1')
    # Create variable object on node 'MyAppl2'.
    currentTimeSlave = variable.Variable(r'MyAppl2/Model Root/slave/Clock/Out1')
```

Related topics

Basics

Basics on Accessing Variables of a Simulation Application on a Remote Node (Real-Time Testing Guide 🕮)
Read/Write Access to Variables of the Simulation Application (Real-Time Testing Guide 🕮)

IsA2L Method

Class

Variable

Syntax

```
OBJ.IsA2L()
```

Purpose

To return whether the variable description file for the real-time application is a TRC or A2L file.

Parameter

–

Return value

The method returns a value of the following type:

Type	Description
Integer	Type of the variable description file: ▪ 0: TRC file ▪ 1: A2L file

Related topics

References

Variable Class..212

Name Method

Class	Variable
Syntax	`OBJ.Name()`
Purpose	To return the name of the simulator variable including subsystems specified in the system description file.
Parameter	–
Return value	The method returns a value of the following type:

Type	Description
String	Name of the simulator variable including subsystems specified in the system description file.

Related topics

References

Variable Class..212

rttlib.watcherlib Module

Introduction

This module provides a class to check conditions according to the ASAM General Expression Syntax (GES) standard.

Where to go from here

Information in this section

Watcher Class........216
To wait in an RTT sequence for the fulfillment of a specified condition.

Operators and Functions Supported by the watcherlib........218
Operators and functions are used in expressions to define the condition for a watcher generator object.

Watch Method........221
To get a watcher generator object that checks the condition in each model step.

Information in other sections

Checking Conditions According to the ASAM GES Standard (Real-Time Testing Guide 📖)
RTT sequences can be paused and resumed depending on conditions that can be specified according to the ASAM General Expression Syntax (GES) standard.

Watcher Class

Syntax

```
from rttlib import watcherlib
WatcherGenerator = watcherlib.Watcher(Condition, \
    VariablesDictionary, Timeout)
```

Purpose

To wait in an RTT sequence for the fulfillment of a specified condition.

Description

A Watcher class lets an RTT sequence wait for the fulfillment of a condition depending on the values of model variables or local variables.

When you create a watcher object, a watcher generator object is returned. When the RTT sequence is executed, it checks the specified condition at the point where the watcher generator object is called. Only if the condition of the watcher object is fulfilled, the following instruction of the RTT sequence is executed.

A watcher object contains all the information needed to check a condition. The condition is specified in a string that contains an expression with variables. You can specify different kinds of expression. For more information on the operators and functions that can be used in the expression, refer to Operators and Functions Supported by the watcherlib on page 218. The variables used in the expression must be variable objects that correspond to model variables. For more information on variable objects, refer to Variable Class on page 212.

The timeout parameter avoids an endless checking. If the condition is not fulfilled and the specified timeout value is reached, a timeout exception is thrown.

Parameter

The class uses the following parameter:

Parameter	Type	Description
Condition	String	Specifies the condition that is checked. To specify a condition, you can use different expressions. Refer to Operators and Functions Supported by the watcherlib on page 218.
VariablesDictionary	Dictionary	Specifies the variables that are used in the condition. It is a dictionary with key-value pairs. The key must be a name that is used in the condition string. The value must be the name of a variable object that corresponds to a model variable.
Timeout	Integer	Specifies a timeout in seconds. If the condition is not fulfilled within the specified time, a timeout exception is thrown.

Attributes

The following attributes are part of the class:

Attribute	Type	Purpose
Condition	String	To get the condition.
Variables	Dictionary	To get the dictionary with the variable names used in the condition.
Timeout	Integer	To get the timeout value.
Description	String	To get or set a description of the watcher.

Methods

The following method is part of the class:

Method	Purpose
Watch	To get a watcher generator object that checks the condition in each model step. Refer to Watch Method on page 221.

Example

The following example shows how to use the watcherlib.

```
from rttlib import watcherlib
from rttlib import variable
if variable.IsA2L:
    BatteryVoltage = variable.Variable(r'BatteryVoltageValue')
else:
    BatteryVoltage = variable.Variable(r'Model Root/BatteryVoltage[V]/Value')
def MainGenerator():
    BatteryVoltage.Value = 12.0
    condition = "var1 > 10"
    timeout = 15
    MyWatcher = watcherlib.Watcher(condition, {'var1':BatteryVoltage}, timeout)
    WatcherGenerator = MyWatcher.Watch()
    # Wait until the condition is true or the timeout is reached.
    yield WatcherGenerator
```

Related topics

Basics

Checking Conditions According to the ASAM GES Standard (Real-Time Testing Guide 🕮)

Operators and Functions Supported by the watcherlib

Introduction

Operators and functions are used in expressions to define the condition for a watcher generator object.

Restrictions

The following restrictions apply to the functions:

- The first parameter of the `posedge`, `negedge`, `changed`, `changedpos`, and `changedneg` functions must be only identifiers, not expressions.
- The number of significant initial characters of an identifier is 32.
- The predefined `INF` and `NaN` constants of the GES must not be used.
- The predefined `epsilon` constant may only be used as expression for the `expr2Delta` parameter of the `changed`, `changedpos`, and `changedneg` function to designate an arbitrary small value unequal to zero.

Supported operators and functions

The following table lists operators and functions that can be used in ASAM XIL API Version 2.0.1 and the watcherlib of Real-Time Testing.

Semantic	Syntax and Arguments	ASAM XIL API V 2.0.1	RTT watcherlib
Sequential evaluation of several trigger conditions: When the left-hand condition evaluates to true, the evaluation of the right-hand condition starts and continues even if the left-hand condition does not remain true	expr1 &> expr2	✓	✓
Conditional operator	expr1 ? expr2 : expr3	–	–
Logical OR	expr1 || expr2	✓	✓
Logical XOR	expr1 ^^ expr2	✓	✓
Logical AND	expr1 && expr2	✓	✓
Bitwise OR (inclusive OR)	expr1 | expr2	–	✓
Bitwise XOR (exclusive OR)	expr1 ^ expr2	–	✓
Bitwise AND	expr1 & expr2	–	✓
Equality; the implementation of a comparison of floating-point numbers is implementation-specific.	expr1 == expr2	✓	✓
Non-equality	expr1 != expr2	✓	✓
Smaller than	expr1 < expr2	✓	✓
Greater than	expr1 > expr2	✓	✓
Smaller or equal	expr1 <= expr2	✓	✓
Greater or equal	expr1 >= expr2	✓	✓
Bitwise shift left, 0 is added at LSB	expr1 << expr2	–	✓
Bitwise shift right, 0 is added at the MSB if MSB was 0 else 1 is added	expr1 >> expr2	–	✓
Addition	expr1 + expr2	✓	✓
Subtraction	expr1 – expr2	✓	✓
Multiplication	expr1 * expr2	✓	✓
Division	expr1 / expr2	✓	✓
Modulo operation	expr1 % expr2	–	✓
Negation	-expr	✓	✓
Positive sign; has no effect, only displays a positive number as in C.	+expr	✓	✓
Logical NOT	! expr	✓	✓
Bitwise NOT	~ expr	–	✓
Postfix operator .	identifier.identifier	–	–
Array element access	identifier[constant]	–	–
Sine (argument in radians)	sin(expr)	✓	✓

Semantic	Syntax and Arguments	ASAM XIL API V 2.0.1	RTT watcherlib
Cosine (argument in radians)	cos(expr)	✓	✓
Tangent (argument in radians)	tan(expr)	–	✓
Arc sine (return value in radians)	asin(expr)	–	✓
Arc cosine (return value in radians)	acos(expr)	–	✓
Arc tangent (return value in radians)	atan(expr)	–	✓
Hyperbolic sine	sinh(expr)	–	✓
Hyperbolic cosine	cosh(expr)	–	✓
Hyperbolic tangent	tanh(expr)	–	✓
Natural logarithm (base e)	log(expr)	–	✓
Common logarithm (base 10)	log10(expr)	–	✓
Exponential function, returns e^{Number}	exp(expr)	–	✓
Power (pow(a,b) ⇒ a^b)	pow(expr1, expr2)	✓	✓
Power operator (a**b) ⇒ a^b)	expr ** expr	✓	–
Square root	sqrt(expr)	–	✓
Absolute value	abs(expr)	✓	✓
Sign (returns -1 for negative number, 0 if zero, +1 for positive number)	sgn(expr)	–	✓
Returns the nearest integer of the given number.	round(expr)	–	–
Returns smallest integer that is greater than or equal to the given number.	ceil(expr)	–	✓
Returns largest integer that is less than or equal to the given number.	floor(expr)	–	✓
Minimum	min(expr1, expr2)	✓	✓
Maximum	max(expr1, expr2)	✓	✓
Detection of positive edge: Returns true if the value of the signal defined by the variable changes from a value smaller than the threshold to a value greater than or equal to the threshold.	posedge(expr1, expr2Threshold)	✓	✓
Detection of negative edge: Returns true if the value of the signal defined by the variable changes from a value higher than the threshold to a value smaller or equal than the threshold.	negedge(expr1, expr2Threshold)	✓	✓
Detection of positive edge: Returns true if the value of the signal defined by the variable changes from a value smaller than the threshold to a value greater than the threshold.	strictposedge(expr1, expr2Threshold)	–	✓

Semantic	Syntax and Arguments	ASAM XIL API V 2.0.1	RTT watcherlib
Detection of negative edge: Returns true if the value of the signal defined by the variable changes from a value higher than the threshold to a value smaller than the threshold.	strictnegedge(expr1, expr2Threshold)	–	✓
Detection of value change: A change is detected if the difference between the current number and its direct successor (number in the last evaluation step) is greater than or equal to the respective delta.	changed(expr1, expr2Delta)	✓	✓
Detection of a positive value change: Returns true when the value of expr1 is increased in relation to the previous evaluation step and the increase is greater than or equal to expr2Delta.	changedpos(expr1, expr2Delta)	✓	✓
Detection of a negative value change: returns true when the value of expr1 is decreased in relation to the previous evaluation step and the decrease is greater than or equal to expr2Delta.	changedneg(expr1, expr2Delta)	✓	✓
Detection of hardware trigger	hwtrigger()	–	–
Detection of manual trigger	mantrigger()	–	–

Related topics

Basics

Checking Conditions According to the ASAM GES Standard (Real-Time Testing Guide 🕮)

Watch Method

Class

Watcher

Syntax

```
WatcherGenerator = Watcher.Watch()
```

Purpose

To get a watcher generator object that checks the condition in each model step.

Parameter

–

Return value

The method returns a value of the following type:

Type	Description
Object	The watcher generator object.

Example

The following shows how to use the method.

```
WatcherGenerator = MyWatcher.Watch()
# Wait until the condition is true or the timeout is reached.
yield WatcherGenerator
```

Related topics

Basics

Checking Conditions According to the ASAM GES Standard (Real-Time Testing Guide 📖)

References

rttlib.watcherlib Module..216
Watcher Class..216

Standard Python Libraries

Where to go from here

Information in this section

Supported Python Modules..223
Shows all the Python modules which are supported by Real-Time Testing.

Supported Methods of the time Module...225
The time module is not fully supported. It depends on the platform which methods you can use.

Supported Python Modules

Introduction

You can use standard Python libraries for your RTT sequences. However, some of the Python functions within these supported modules are not suitable for Real-Time Testing as they have non-deterministic execution times. The `sort()` function, for example, can be used for small data sets, but can cause long and non-deterministic execution times for larger data sets.

Supported Python modules

The following table shows all the Python modules which are supported by Real-Time Testing. For a description of the modules and their functions, refer to the *Python Reference*, which is available at http://www.python.org.

Functional Group	Module Name	Module Description
Python Runtime Services	sys	System-specific parameters and functions
	gc	Garbage Collector interface
	types	Names for built-in types
	operator	Standard operators as functions
	traceback	Print or retrieve a stack traceback
	_pickle	A faster pickle
	copyreg	Register pickle support functions
	copy	Shallow and deep copy operations
	marshal	Internal Python object serialization
	warnings	Warning control
	builtins	Built-in objects
	__main__	Top-level script environment
	__future__	Future statement definitions
String Services	re	Regular expression operations
	struct	Interpret strings as packed binary data
	codecs	Codec registry and base classes
	encodings ▪ ascii ▪ cp1252 ▪ p932 ▪ latin_1 ▪ utf_8	Standard encodings
Miscellaneous Services	math	Mathematical functions
	cmath	Mathematical functions for complex numbers
	random	Generate pseudo-random numbers
	bisect	Array bisection algorithm
	collections	High-performance container data types
	heapq	Heap queue algorithm
	array	Efficient arrays of numeric values
	sets	Unordered collections of unique elements
	itertools	Functions creating iterators for efficient looping
Generic Operating System Services	os	Miscellaneous operating system interfaces `os.times` is not supported
	errno	Standard errno system symbols
	time	Time access and conversion (supported by Real-Time Testing 3.2 and later) The time module is not fully supported. It depends on the platform type which methods you can use. Refer to Supported Methods of the time Module on page 225.
Internet Data Handling	binascii	Convert between binary and ASCII

Related topics

Basics

Using Modules from the Standard Python Library (Real-Time Testing Guide 🕮)

Supported Methods of the time Module

Introduction

The time module is not fully supported. It depends on the platform type which methods you can use.

Supported/unsupported methods

The following table shows you which methods of the time module are supported on the platforms.

Method	Platform						
	DS1006 Processor Board	DS1007 PPC Processor Board	MicroAutoBox	MicroLabBox	SCALEXIO Processing Unit	DS6001 Processor Board	VEOS
`time.time()`	✓	✓	✓	✓	✓	✓	✓
`time.clock()`	✓	✓	✓	✓	✓	✓	✓
`time.gmtime([secs])`	–	–	–	–	✓	✓	✓
`time.localtime([secs])`	–	–	–	–	✓	✓	✓
`time.asctime([t])`	–	–	–	–	✓	✓	✓
`time.ctime([secs])`	–	–	–	–	✓	✓	✓
`time.strftime(format[, t])`	–	–	–	–	✓	✓	✓
`time.sleep(secs)`	–	–	–	–	–	–	–
`time.mktime(t)`	–	–	–	–	–	–	–
`time.strptime(strin[, format])`	–	–	–	–	–	–	–
`time.tzset()`	–	–	–	–	–	–	–

Related topics

References

Supported Python Modules..223

dSPACE.Common.MessageHandler.Logging Reference

Where to go from here

Information in this section

ILogMessage Interface..227
To access information about a message as written to a log file.

ILogSession Interface...229
To access information about a message log session.

MessageReader Class...230
To read serialized messages written by dSPACE products.

MessageReaderSettings Class...232
To define the settings of a message reader.

Severity Enumeration...233
To specify the severity of a message.

ILogMessage Interface

Namespace `dSPACE.Common.MessageHandler.Logging`

Description To access information about a message as written to a log file.

Properties

The element has the following properties:

Name	Description	Get/Set	Type
IsStartMessage	Gets a value indicating whether the message is a session start message.	Get	*Boolean*
IsStopMessage	Gets a value indicating whether the message is a session stop message.	Get	*Boolean*
MainModuleNumber	Gets the main module number of the message.	Get	*Integer*
MessageCode	Gets the error code of the message.	Get	*Integer*
MessageText	Gets the text of the message.	Get	*String*
ModuleName	Gets the module name of the message.	Get	*String*
Session	Gets the log session which issued the message.	Get	ILogSession (refer to ILogSession Interface on page 229)
Severity	Gets the severity of the message.	Get	Severity (refer to Severity Enumeration on page 233)
SubmoduleNumber	Gets the submodule number of the message.	Get	*Integer*
ThreadId	Gets the thread ID of the submitting thread.	Get	*Integer*
TimeStamp	Gets the time when the message was submitted. Given as local time in the time zone of the session.	Get	*DateTime*
UtcTimeStamp	Gets the time when the message was submitted in UTC time.	Get	*DateTime*

Methods

The element has no methods.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API (Real-Time Testing Guide 🕮)

Examples

Example of Reading Messages with C# (Real-Time Testing Guide 🕮)
Example of Reading Messages with Python (Real-Time Testing Guide 🕮)

References

ILogSession Interface..229
Severity Enumeration..233

ILogSession Interface

Namespace `dSPACE.Common.MessageHandler.Logging`

Description To access information about a message log session.

Properties The element has the following properties:

Name	Description	Get/Set	Type
CloseTime	Gets the time when the session was closed. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. Given as local time in the time zone of the session.	Get	*DateTime*
IsOpen	Gets a value indicating whether the session is still open. If true, the session is still open and new messages can be written.	Get	*Boolean*
IsValid	Gets a value indicating whether the session is valid. A session can become invalid if its log files are corrupted.	Get	*Boolean*
MetaData	Gets the products metadata as read from log file session info.	Get	*Dictionary< String, String >*
ProcessId	Gets the process ID of the log session.	Get	*Integer*
ProductName	Gets the product name of the log session.	Get	*String*
SessionId	Gets the ID of the log session. This ID is unique in the context of its session reader.	Get	*Integer*
StartTime	Gets the sessions start time. Given as local time in the time zone of the session.	Get	*DateTime*
TimezoneName	Gets the standard time zone name of the session.	Get	*String*
TimezoneOffset	Gets the time zone offset of the session relative to UTC.	Get	*TimeSpan*
UtcCloseTime	Gets the time when the session was closed as UTC time. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully.	Get	*DateTime*
UtcStartTime	Gets the start time of the log session as UTC time.	Get	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
ToSessionTime	Converts UTC time to time zone used when the session was written.	▪ *<DateTime>* `utcTime`: Specifies the UTC time to convert.	Time in the time zone of the logging session. ▪ DateTime

[1] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API (Real-Time Testing Guide 🕮)

Examples

Example of Reading Messages with C# (Real-Time Testing Guide 🕮)
Example of Reading Messages with Python (Real-Time Testing Guide 🕮)

MessageReader Class

Description

To read serialized messages written by dSPACE products.

Constructor

The element has the following constructor:

Name	Description	Parameter[1]	Returns
MessageReader	Initializes a new instance of the MessageReader class.	▪ <MessageReaderSettings>[2] `settings`: Settings which allow to specify which sessions and messages are read. Can be null, causing all existing log files to be read.	None

[1] <Type> `Name`: Description

[2] Refer to MessageReaderSettings Class on page 232

Properties

The element has no properties.

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
Dispose	Performs application-specific tasks associated with freeing, releasing, or resetting unmanaged resources.	None	None
ReadMessages	Reads the messages written to the log files of the sessions up to now. The messages are returned in chronological order according to their time stamps. **Note** The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time.	None	Messages read from log file. ▪ IEnumerable< ILogMessage (refer to ILogMessage Interface on page 227) >

[1] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API (Real-Time Testing Guide 🕮)

Examples

Example of Reading Messages with C# (Real-Time Testing Guide 🕮)
Example of Reading Messages with Python (Real-Time Testing Guide 🕮)

References

MessageReaderSettings Class..232

MessageReaderSettings Class

Description

To define the settings of a message reader.

Used to filter the log sessions and messages read.

Constructor

The element has the following constructor:

Name	Description	Parameter[1]	Returns
MessageReaderSettings	Initializes a new instance of the MessageReaderSettings class.	None	None

[1] <Type> `Name`: Description

Properties

The element has the following properties:

Name	Description	Get/Set	Type
DirectoryNames	Gets a list of specific directory names from which to read log files. If the list is empty, all standard directories are searched for log files.	Get	*List< String >*
MaximalSessionCount	Gets or sets the maximal number of log sessions read for each product. If the count is a positive number n, only the last n sessions are read. If the count is not positive, an unlimited number of sessions is read. The default value is zero, i.e., unlimited.	Get/Set	*Integer*
MessageTimeAfter	Gets or sets the minimal time for which messages are read, given as UTC time. Only messages submitted after the message time are read. The message time may be in the past. The message time must be given as valid UTC time. The default time is undefined, i.e., each message time is allowed.	Get/Set	*DateTime*
Products	Gets the list of product names for which to read log sessions. If the list is empty sessions of all products are read.	Get	*List< String >*
StartTimeAfter	Gets or sets the minimal start time for which sessions are read, given as UTC time. Only sessions which started after the start time are read. The start time may be in the past. The start time must be given as valid UTC time. The default time is undefined, i.e., each start time is allowed.	Get/Set	*DateTime*

Methods

The element has the following methods:

Name	Description	Parameter[1]	Returns
SetDirectoryNames	Sets the list of specific directory names from which to read log files. You do not have to specify a list. If the list is empty, all standard directories are searched for log files.	*<string[]>* `names`: Array of directory names.	None
SetProducts	Sets the list of product names for which to read log sessions.	*<string[]>* `products`: Array of product names.	None

[1] <Type> `Name`: Description

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API (Real-Time Testing Guide 📖)

Examples

Example of Reading Messages with C# (Real-Time Testing Guide 📖)
Example of Reading Messages with Python (Real-Time Testing Guide 📖)

Severity Enumeration

Description

To specify the severity of a message.

Enumeration values

The enumeration has the following values:

Value	Name	Description
0	Trace	A trace message. Trace messages are usually not created. It depends on the host application if it is possible to configure the message handler to create trace messages.
1	Info	An information message.
2	Warning	A warning message.
3	Error	An error message.
4	SevereError	A severe error message.
5	SystemError	A system error message.

Value	Name	Description
6	Question	A question message.
7	Advice	An advice message.

Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API (Real-Time Testing Guide 📖)

Examples

Example of Reading Messages with C# (Real-Time Testing Guide 📖)
Example of Reading Messages with Python (Real-Time Testing Guide 📖)

A

AccessBoard 49
accessing
  platform 49
AccessProviderInfo class 166
ActivateInterface method 169
AddVariable (datastream) 144, 153
ApplicationRTTVersion 50
ASAM General Expression Syntax 216
ASM GES 216

B

BCGServiceProvider 52
Board class 54

C

canmmbaselib 94
canmmerror 110
canmmlib 96
channel 100
circular references 208
Close (rs232lib) 196
commands
  dSPACE Help 15
common constants
  rttmanagerlib 84
Common Program Data folder 12
Connect
  Real-Time Test Manager 21
Continue 70
  Real-Time Test Manager 31
ContinueAll 57
controller 98
controllers 94
Copy Error Text
  Real-Time Test Manager 32
Create 57
  Real-Time Test Manager 22
CreateVariableMap (datastream) 142
CreateVariableMapMDF (datastream) 150

D

datastream
  AddVariable 144, 153
  CreateVariableMap 142
  CreateVariableMapMDF 150
  MatFile 146
  MDFFile 154
  Replay 149, 158
DataStreams 77
DeactivateInterface method 170
Delete
  Real-Time Test Manager 32
Documents folder 12
dscanapilib class
  ActivateChannel 115
  BusStatistics 136
  DeactivateChannel 116
  EnableBusStatistics 116
  EncodeBusStatistics 117
  FlushReceiveQueue 118
  FlushTransmitQueue 119
  GetAvailableChannels 120
  GetBaudrate 120
  GetBusInfo 121
  GetBusType 122
  GetErrorText 123
  GetHardwareTime 123
  GetHardwareTimeResolution 124
  InitChannel 125
  ReadReceiveQueue 126
  RegisterChannel 127
  ResetHardwareTime 128
  SetAcceptance 129
  SetBaudrate 130
  SetChannelOutput 131
  SetTransmitAcknowledge 132
  TransmitMessages 132
  UnregisterChannel 133
dSPACE log
  printing message 204
DynamicVariable 160

E

errors 163
EthRawFrame class 168
EthRawFrameHeader class 167
ExecutionError 78
Exit
  Real-Time Test Manager 16

F

FlushReceiveQueue method 171
FlushTransmitQueue method 172
ftEXT 94
ftStd 94

G

Garbage Collector 208
General Properties 40
Generate 53
GetAccessProvidersCount method 173
GetAvailableInterfaces method 173
GetChannel 99
GetController 96
GetInterfaceCapabilities method 175
GetInterfaceMacAddress method 175
GetNumInBytes (rs232lib) 192
GetRawMessage 101
GetSequenceArgument 202
GetSupportedFrameTypes method 176
GetSupportedFrameTypesCount method 177
GetTime method 178
GetTimeResolution method 178
GetTraceBackString
  rttutilities 86

H

Hostcall 188

I

InterfaceInfo class 166
IsA2L 214
IsInterfaceAccessible method 181
IsInterfaceAvailable method 181
Item (Sequences) 59
Item (Variables) 80

L

Local Program Data folder 12
Logging class 204

M

MatFile (datastream) 146
MDFFile (datastream) 154
memory leak 208
message 102
  printing 204
messagerx 107
messagetx 108

N

Name 215
Name (DynamicVariable) 161
New Features and Migration command 18

O

OnAdd 82
OnCreate 65
OnError 65
OnHostCall 76
OnRemove 66
OnStateChanged 67
OnWrite 67
Open
  Real-Time Test Manager 33
OpenEx (rs232lib) 190

P

Parallel 198
ParallelRace 199
Pause 71
  Real-Time Test Manager 34
PauseAll 60
platform
  accessing 49
PurgeComm (rs232lib) 196
Python Libraries
  standard 223
Python modules
  standard
    supported 223

R

Read (rs232lib) 193
ReadFrames method 182
Real-Time Test Manager
  Connect 21
  Continue 31
  Copy Error Text 32
  Create 22
  Delete 32
  Exit 16
  Explore Demos Folder 39
  General Properties 40
  Open 33
  Pause 34
  Refresh Platform Configuration 25
  Reload 34
  Run 35
  Stop 37
  Unzip Demos 41
RealTimeTestManagerServer 48
Receive 104
ReceiveGen 105
Refresh Platform Configuration
  Real-Time Test Manager 25
RegisterInterface method 183
Reload
  Real-Time Test Manager 34
Remove 60, 71
Replay (datastream) 149, 158
RTT version
  reading 50
RTTException
  errors 163
rttlib.canlib 91
rttlib.datastream 141
rttlib.dsethernetapilib 164
rttlib.dynamicvariable 160
rttlib.globalvariables 187
rttlib.hostcall 188
rttlib.rs232lib 190
rttlib.scheduler 198
rttlib.utilities 202
rttmanagerlib 44
  common constants 84
  overview 45
RTTSleep
  rttutilities 87
rttutilities 86
Run 72
  Real-Time Test Manager 35
RunAll 61

S

Sequence 69
SequenceEvents 74
SequenceProperties 206
Sequences 56
SequencesEvents 63
SetCallGCAfterRemovingAllSequences 208
SetConfig (rs232lib) 191
SetFilter method 184
SetImportTimeslice 209
standard Python modules
  supported 223
Stop 73
  Real-Time Test Manager 37
StopAll 61

T

Transmit 106
TransmitFrames method 185
TransmitGen 106

U

UnregisterInterface method 186
Unzip Demos
  Real-Time Test Manager 39, 41

V

Variable 83, 212
Variables 79
VariablesEvents 81

W

Wait 211
Watch 221
Watcher 216
Write (rs232lib) 194
WriteString (rs232lib) 195