RT Box

DEMO MODEL

Automated Testing

Using the Robot framework with Python and PLECS and the RT Box over XML-RPC

Last updated in RT Box TSP 2.0.4

www.plexim.com

- Request a PLECS and PLECS Coder trial license
- Get the latest RT Box Target Support Package
- Check the PLECS and RT Box documentation

1 Introduction

For automated test environments the RT Box can be controlled via external scripts using an XML-RPC interface. More information on the RT Box's XML-RPC interface can be found in the RT Box User Manual [1]. XML-RPC is a lightweight protocol for executing functions on a remote machine. The RT Box acts as an XML-RPC server, which processes requests sent from scripts running on another computer. Many scripting languages support XML-RPC out of the box, for example Python. For test automation the XML-RPC interface can be used together with the open-source automation framework "Robot".

This demo shows how to set up a basic automated test for the RT Box by using the XML-RPC interface of the RT Box and the Robot Framework.

1.1 Required Software and Hardware

The following section lists the needed software and hardware to fully run this automated test demo example. Make sure to enable the XML-RPC interface in PLECS under **File + PLECS Preferences... + General** by checking the box **XML-RPC interface** and setting the port to 1080.

Source Files

There are three source files included with this model: a PLECS model (`.plecs`) and two `.py` Python files. Theses files can be found in the RT Box Demo Models section of the PLECS Help Viewer.

Installing Python

For Windows and Mac operating systems, Python 3.x can be newly installed or updated from `https://www.python.org/downloads/`.

Installing matplotlib and numpy

Install `matplotlib` by entering the following commands:

For Python 3.x:

- Using the Windows Command Prompt:

  ```
  py -3 -m pip install -U matplotlib
  ```

  If you receive an error message that the `pip` module is unknown, you will need to install it first, before installing `matplotlib`.
- Using the Mac Terminal:

  ```
  python3 -m pip install -U matplotlib
  ```

You can use the same approach for the installation of the `numpy` package.

Installing Robot Framework

The Robot Framework is installed by using `pip`. For Python 3.x

- Using the Windows Command Prompt:

  ```
  py -3 -m pip install -U robotframework
  ```

- Using the Mac Terminal:

  ```
  python3 -m pip install -U robotframework
  ```

Further installation instructions can be found in the **Installation instruction** chapter in the official Robot Framework User Guide [2].

Hardware

An RT Box (any version) is connected to the host computer via Ethernet is required. In addition, valid PLECS and PLECS Coder licenses are needed. To request a trial license for these products, please visit www.plexim.com/trial. Furthermore, you need a current Target Support Package of the RT Box and two 37 pin Sub-D cables to connect the Analog Out interface with the Analog In interface, and Digital Out interface with Digital In interface.

2 Robot Framework

The Robot Framework is open and extensible and can be integrated with virtually any other tool to create powerful and flexible automation solutions. Being open source also means that Robot Framework is free to use without licensing costs. Robot Framework utilizes the keyword-driven testing approach. Its capabilities can be extended by libraries implemented in Python. The Robot Framework is hosted on GitHub where you can find further documentation, source code, and issue tracker. It runs independent of the operating system and the core framework is implemented in Python.

The Robot Framework has a modular architecture that can be extended with user defined libraries. Data is defined in files using the syntax shown in the example below. A file containing a set of tests creates a suite. When executing a robot test, the framework first parses the data. It then utilizes keywords to interact with the target system, i.e. the PLECS RT Box. Robot tests can be started from the command line. As a test result you get a report and log in HTML format as well as an XML output. These provide a complete record of the system behavior during the test. An overview of the available keywords specific to the RT Box is given in section 7.

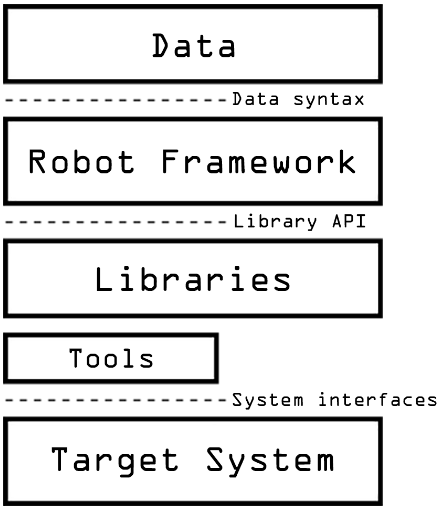


Figure 1: Modular architecture of the Robot Framework.

3 Model

The model used for demonstrating an automated test with the Robot Framework is a basic boost converter, as explained in the RT Box Demo Model **Boost Converter**. This model has been expanded with specific blocks for the automated test, i.e. a Programmable Value block, Data Capture blocks and a UDP Send block.

3.1 Controller

The inductor current of the boost converter is regulated by a PI controller. The controller receives its set point from a Programmable value block. Until the block receives a new value via XML-RPC, the initial value is at its output. When a new value is received by the Programmable Value block its output **v** changes from zero to one. This triggers the data capturing of the three Data Capture blocks. Once the real-time simulation is started, the UDP Send block sends data over the network to the host computer specified by its IP address.

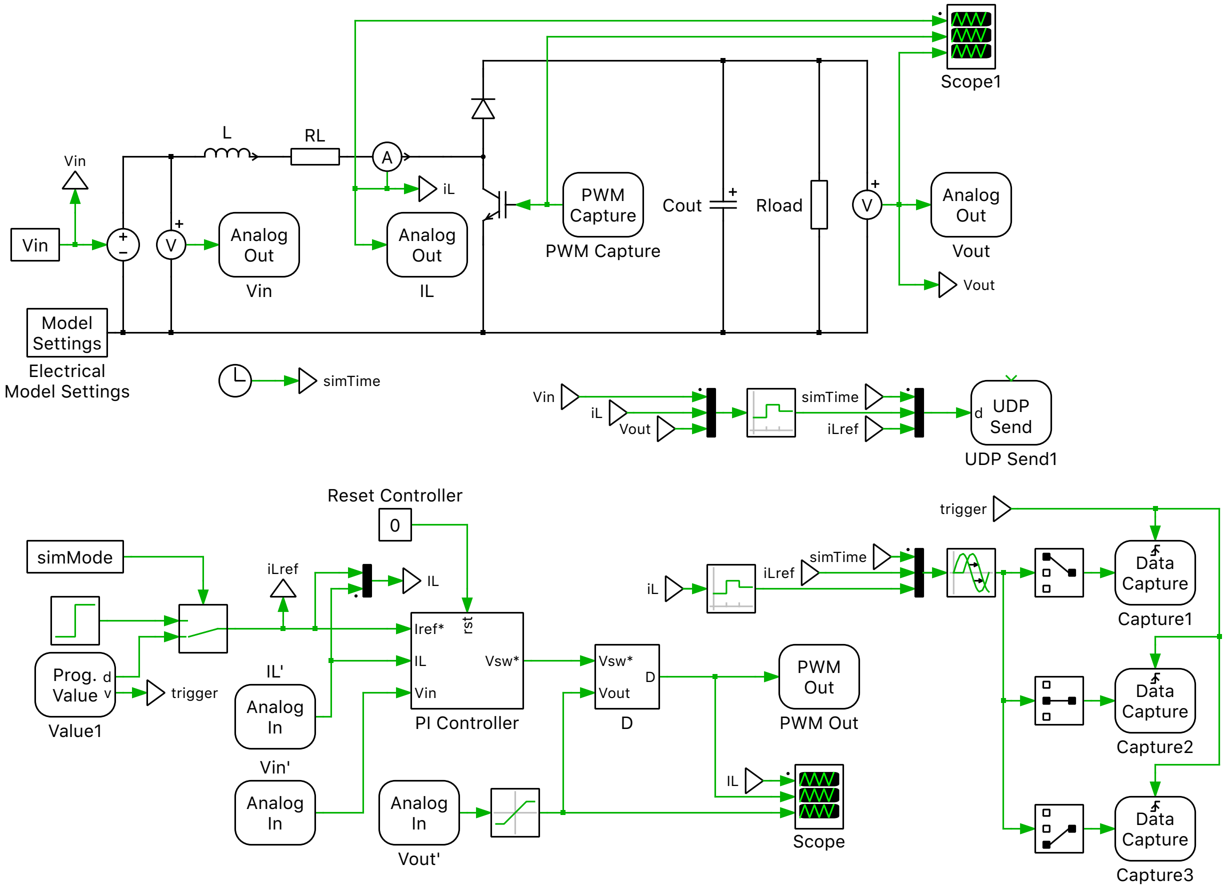


Figure 2: Boost converter plant and controller model.

4 Automated Test with Robot Framework

In this section the keyword driven test for the boost converter from section 3 is explained. A Robot Framework data file is divided into different sections as listed below:

- Variables: Defining variables that can be used elsewhere in the test data.
- Settings: Importing self-made test libraries.
- Keywords: Creating user keywords from existing low-level keywords.
- Test Cases: Creating test cases from existing keywords (low-level or defined in section “Keywords”).

The sections mentioned above are identified by their respective header row, e.g. `*** Variables ***`.

4.1 Variables

In the “Variables” section one can define variables that can then be used elsewhere in the test data file. In this demo the RT Box host name, the IP address of the host computer, the sample time of the UDP block, the number of signals send over UDP and the header of the created .csv file as well as the model name are defined as variables.

```
*** Variables ***
${RTBOX}=  RTBox.local.
${HOST_PC_IP_ADDRESS}=  10.0.0.216
${UDP_SAMPLE_TIME}=  ${0.001}
${UDP_signal_width}=  ${5}
${UDP_HEADER}=  [time, Vin (V), Vout (V), iL (A), iLref (A)]
${MODEL_NAME}=  boost_converter
```

4.2 Settings

The “Settings” section is used to import test libraries that contain specific low-level keywords. In this example the RT Box specific library has to be imported by using the physical path to the library file. The path is always relative to the directory where the current test data file is situated. In addition, a second library (`boost_converter.py`) with model specific keywords is imported.

```
*** Settings ***
Library  ./PlecsXMLRPC.py
Library  ./boost_converter.py
```

Note Further project specific low-level keywords can be added in a separate library. This library can then be imported in the “Settings” section.

4.3 Keywords

The `Keywords` section is used to define new keywords based on already existing low-level keywords. This is useful to group several subtasks into bigger task. In this demo, the keyword `Compile and upload` is formed based on the existing low-level keywords `rtBoxSetupServer`, `plecsGenerateCode` and `rtBoxUpload`.

```
*** Keywords ***
Compile and upload
   [Arguments]    ${modelName}  ${subsystemName}  &{codegenVars}
   rtBoxSetupServer  ${RTBOX}
   plecsGenerateCode        ${modelName}   ${subsystemName}   &{codegenVars}
   rtBoxUpload    ${CURDIR}${/}${modelName}   ${subsystemName}
```

Therefore, by executing the new keyword `Compile and upload` first the XML-RPC connection to the RT Box is initialized, then code is generated from the specified model and once this is done, the .elf file is automatically uploaded to the RT box defined by its host name.

4.4 Test Cases

The different test cases defined in this section are always executed one-by-one and execution starts from the top-level test case. Normally the execution of the current test case ends if any of the keywords fails or if all keywords in the test case are run in a sequence. In this demo, the following test sequence is executed:

```
*** Test Cases ***
Load profile and Data Logging
    plecsLoadModel      ${MODEL_NAME}
        Compile and upload  ${MODEL_NAME}  Plant_and_Controller
    ...  Ts_udp=${UDP_SAMPLE_TIME}  host_PC_IP_address=${HOST_PC_IP_ADDRESS}
    ...  simMode=${0}
    plecsGetCircuitBitmap  Plant_and_Controller
    rtBoxBackgroundLogging  ${True}  ${UDP_signal_width}  ${UDP_SAMPLE_TIME}
    ...  background_log.csv  ${UDP_HEADER}
    ${value}=  rtBoxBackgroundLoggingEnabled
    rtBoxStart
    sleep  2
    rtBoxSetValue Value1  90.0
    ${time}=  rtBoxGetValue   Capture1
    ${iLref}=  rtBoxGetValue   Capture2
    ${iL}=  rtBoxGetValue   Capture3
    plotData  ${time}  ${iLref}  iLref  time (s)  current (A)
    ...  current step response  True  -  r
    plotData  ${time}  ${iL}  iL  time (s)  current (A)
    ...  current step response  False  -  b
    rtBoxBackgroundLogging  ${False}
    rtBoxStop
```

- First, the model is opened, compiled and uploaded on the RT box defined in the variable `${RTBOX}`. With the optional *'optStruct'* argument of the `plecsGenerateCode` keyword different values are passed to the respective parameters in the model file.
- Afterwards, by using the `plecsGetCircuitBitmap` keyword, a bitmap of the actual circuit running on the RT Box is logged to the report created at the end of the automated test.
- Before the real-time simulation on the RT Box is started, the data logging over UDP is enabled. This is achieved by setting the `rtBoxBackgroundLogging` keyword to `True` and initializing it with the correct parameters.
- Once the background logging is enabled, the real-time simulation on the RT Box is started by using the `rtBoxStart` keyword.
- After a pause of 2 seconds, the output value of the Programmable Value block with the name "Value1" is changed to 90.0. This triggers the data capturing of the three Data Capture blocks in the model.
- The captured data is read with the `rtBoxGetValue` keyword.
- Once the data is received on the host computer, it is plotted and appended to the test report.
- Finally, the UDP background logging is disabled and the simulation on the RT Box stopped.

4.5 Launch a Robot Test File

The easiest way to execute of a Robot Framework test is by using a console in the working directory. A single file can be executed by typing the following line in the console:

```
robot filename.robot
```

where "filename" is referring to the actual name of the *.robot* file that you want to run.

5 Simulation

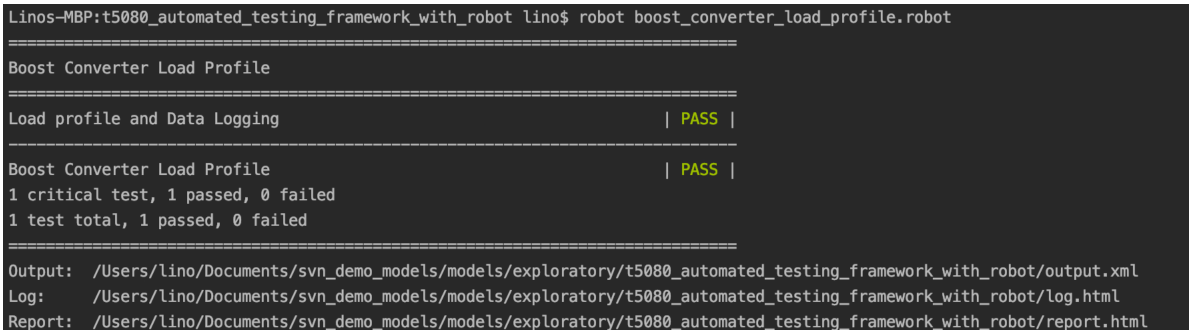
Before launching the test, modify the variables (`${RTBOX}` and `${HOST_PC_IP_ADDRESS}`) in the `boost_converter.robot` to match your setup. Modify the file by using an suitable text editor. The variable `${RTBOX}` refers to the host name of your RT Box. The variable `${HOST_PC_IP_ADDRESS}` has to match the IP address of your computer running PLECS and the Robot Framework.

```
*** Variables ***
${RTBOX}=  RTBox.local.
${HOST_PC_IP_ADDRESS}=  255.255.255.255
```

Before you launch the Robot Framework test, you have to start PLECS on your computer. Once PLECS has started, the automated test is executed by typing

```
robot boost_converter.robot
```

in the console. Please note, that the terminal has to be started on the current working directory. During the test, progress is indicated by dots every time the execution of a keyword is finished successfully. Once the test is finished either "Pass" or "Fail" is displayed in the console for every test case and a log file and a HTML report file are created in the current working directory. The `report.html` file can be opened by your standard web browser. Browse in the report to see the results of the individual keywords. The keyword `rtBoxBackgroundLogging` has created a `.csv` file in the working directory. This file contains the signal values of the received UDP data evenly spaced by the used sample time. A plot with the captured data of the three Data Capture blocks is appended to the report.html file, as shown in Fig. 4.



```
Linos-MBP:t5080_automated_testing_framework_with_robot lino$ robot boost_converter_load_profile.robot
==============================================================================
Boost Converter Load Profile
==============================================================================
Load profile and Data Logging                                         | PASS |
------------------------------------------------------------------------------
Boost Converter Load Profile                                          | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /Users/lino/Documents/svn_demo_models/models/exploratory/t5080_automated_testing_framework_with_robot/output.xml
Log:     /Users/lino/Documents/svn_demo_models/models/exploratory/t5080_automated_testing_framework_with_robot/log.html
Report:  /Users/lino/Documents/svn_demo_models/models/exploratory/t5080_automated_testing_framework_with_robot/report.html
```

Figure 3: Output in the console after the automated test.

6 Conclusion

This demo model demonstrates how to run an automated test with the Robot Framework on the RT Box. During an automated test, all relevant signals can be logged to the host computer by using UDP initiated by the `rtBoxBackgroundLogging` keyword. After the test run, a comprehensive report file is created automatically by the Robot Framework. You can open this test report with any web browser.

7 Robot Framework Library for the PLECS RT Box

Libraries provide the actual automation and testing capabilities to Robot Framework by providing low-level keywords. Several standard libraries are bundled with the framework, and there are separately developed external libraries to interact with the dedicated target system, i.e. the PLECS RT Box. Hereafter, an overview of the available keywords to interact with the PLECS RT box is shown. All arguments given in *italic* are optional.

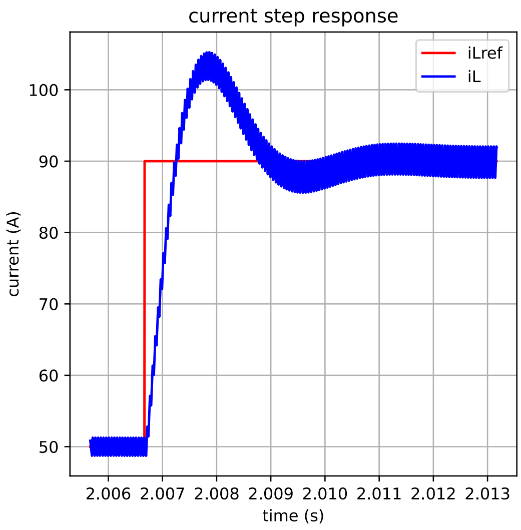


Figure 4: System response to current reference step captured on the RT Box.

Overview of XML-RPC commands

Keyword	Arguments	Return Value
`plecsLoadModel`	`path`	
`plecsGenerateCode`	`modelname`, *`subsystemName`*, *`rtboxType`*, *`optStruct`*	
`plecsGetCircuitBitmap`	`subsystemName`	
`rtBoxSetupServer`	`rtBoxName`	
`rtBoxStart`	*`waitForTrigger`*	
`rtBoxStop`		
`rtBoxUpload`	`modelName`, *`subsystemName`*	
`rtBoxReboot`		
`rtBoxGetStatus`	*`identifier`*	struct
`rtBoxGetVersion`	*`identifier`*	struct
`rtBoxGetHostname`		string
`rtBoxQuerySimulation`	*`identifier`*	struct
`rtBoxQueryCounter`	*`identifier`*	struct
`rtBoxResetCounter`		
`rtBoxGetLEDs`		list
`rtBoxSetValue`	`path`, `value`	
`rtBoxGetValue`	`path` *`triggerCount`*	struct
`rtBoxCaptureTriggerCount`	`path`	
`rtBoxWaitForTrigger`	`path`, `timeout`	

`rtBoxIsRunning`		bool
`rtBoxGetDataCaptureBlocks`		list
`rtBoxGetProgrammableValueBlocks`		list
`rtBoxBackgroundLogging`	`enable`, *signalWidth*, *sampleTime*, *fileName*	
`rtBoxBackgroundloggingEnabled`		

plecsLoadModel ’path’

opens the model defined by its relative path in respect to the current working directory.

plecsGenerateCode ’modelName’ *’subsystemName’ ’rtboxType’ ’optStruct’*

initiates the code generation of the model defined by the *modelName* and optionally the *subsystemName*. The optional argument *rtboxType* can be used to define a target different to RT Box 1. Valid entries for this argument are: `RT Box 1`, `RT Box 2`, `RT Box 3`. The struct *optStruct* is used to generate code for different parameter values without having to modify the model file.

plecsGetCircuitBitmap *’subsystemName’*

adds a bitmap of the model to the test report automatically generated by the Robot Framework. Optionally, also a bitmap of a circuit in a subsystem can be created by providing the *subsystemName*.

rtBoxSetupServer ’rtBoxName’

establishes an XML-RPC communication to the RT Box defined by its hostname.

rtBoxStart *’waitForTrigger’*

starts the execution of the real-time simulation on the RT Box. With the optional argument *waitForTrigger* the start of the real-time simulation can be delayed until it receives the start-trigger over SFP. The *waitForTrigger* argument is only useful if the startup is synchronized over SFP with another RT Box.

rtBoxStop

stops the execution of the real-time simulation on the RT Box.

rtBoxUpload ’modelName’ *’subsystemName’*

loads a executable (ELF file) on the RT Box. The `.elf` file can be generated by using the `plecsGenerateCode` keyword explained above.

rtBoxReboot

reboots the RT Box. During the reboot-process, the RT Box will be non-responding.

`rtBoxGetStatus` ***`'identifier'`***

returns a struct containing the RT Box temperature, fan speed, the application log and the model timestamp. With the optional argument *identifier*, one can specify a single return value out of the struct. Valid values for the *identifier* argument are:

```
'temperature','fanSpeed','logPosition','applicationLog','clearLog','modelTimeStamp'
```

`rtBoxGetVersion` ***`'identifier'`***

returns a struct containing the RT Box serial number, the RT Box revision version, the CPU serial number, the build number of the firmware running on the RT Box including the date of the build, the firmware version, the FPGA version number, the actual IP address and the RT Box MAC address. With the optional argument *identifier* one can specify a single return value out of the struct. Valid values for the *identifier* are:

```
'board','boardRevision','cpu','firmwareBuild','firmwareVersion','fpgaVersion',
'ipAddresses','mac'
```

`rtBoxGetHostname`

returns the name of the RT Box as a string.

`rtBoxQuerySimulation` ***`'identifier'`***

returns a struct containing the version number of the RT Box target support package, the name of the model running on the RT Box, the actual model time stamp, and the discretization step size of the model running on the RT Box. With the optional argument *identifier* one can specify a single return value out of the struct. Valid values for the *identifier* are:

```
'applicationVersion','modelName','modelTimeStamp','sampleTime'
```

`rtBoxQueryCounter` ***`'identifier'`***

returns a struct containing the maximum cycle time of the model running on the RT Box, the model time stamp and the current cycle time. With the optional argument *identifier* one can specify a single return value out of the struct. Valid values for the *identifier* on an RT Box 1 are:

```
'maxCycleTime','modelTimeStamp','runningCycleTime'
```

For RT Box 2 and 3, the following values are permissible:

```
'maxCycleTime1','maxCycleTime2','maxCycleTime3','modelTimeStamp',
'runningCycleTime1','runningCycleTime2','runningCycleTime3'
```

The index number (1, 2 and 3) refer to the respective core on the RT Box 2 and 3.

`rtBoxResetCounter`

resets the **maximum cycle time** value on the RT Box.

`rtBoxGetLEDs`

returns a struct containing the status of the four LEDs on the front panel of the RT Box 1. For each LED the return value is either '1' if the LED is lit or ' ' if the LED is off. This keyword can only be used on the RT Box 1.

`rtBoxSetValue 'path' 'value'`

sets the output of the Programmable Value block indicated by *path*. The parameter *path* is the path of the block relative to the code generation subsystem. The parameter *value* must be an XML-RPC array where the number of elements corresponds to the width of the output signal.

`rtBoxGetValue 'path'` ***`'triggerCount'`***

returns the last filled data buffer from the Data Capture block indicated by *path*. IF specified, the keyword waits until the sample buffer has been filled the amount of times given in the optional argument *triggerCount*.

`rtBoxCaptureTriggerCount 'path'`

returns how many times the sample buffer of the Data Capture block indicated by *path* has been filled.

`rtBoxWaitForTrigger`

`rtBoxIsRunning`

returns `True` if the real-time simulation is running on the RT Box.

`rtBoxGetDataCaptureBlocks`

returns a list of all Data Capture blocks in the current model, with path.

`rtBoxGetProgrammableValueBlocks`

returns a list of all Programmable Value blocks in the current model, with path.

`rtBoxBackgroundLogging 'enable'` ***`'signalWidth' 'sampleTime' 'fileName'`***

enables the background logging during an automated test with the Robot Framework if the argument `enable` is set to `True`. The background logging keyword starts a background thread that sets up a UDP socket that is listening on port **52345**. In order to make the background logging work, there must be at least one UDP block in the model file. In addition, the remote IP address configured in the UDP send block has to match the IP address of the host PC. Always use the port number **52345** for the background logger. The argument *signalWidth* has to match the number of signals connected to the UDP send block in the PLECS model. The same is true for the argument *sampleTime*. The fastest allowed sample time is 1 ms. If the keyword is used to disable the background logger by setting the argument *enable* to `False`, all other arguments are not required.

`rtBoxBackgroundloggingEnabled`

returns `True` if the background logging is enabled. If not, the keyword returns `False`.

References

[1] *RT Box User Manual*, Plexim GmbH, Online: https://www.plexim.com/sites/default/files/rtboxmanual.pdf

[2] Robot Framework Foundation, Robot Framework User Guide, Version 3.2.1, [Online]. Available: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html. [Accessed: June. 12, 2020].

Revision History:

RT Box TSP 2.0.4 First release

How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	`info@plexim.com`	Email
	`http://www.plexim.com`	Web

RT Box Demo Model