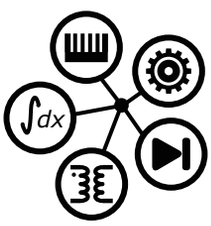


RT Box
DEMO MODEL

Automated Testing

Python、PLECSおよびRT BoxをXML/JSON-RPCを介してRobot Frameworkを使用する

Last updated in RT Box TSP 3.1.5

KESCO KEISOKU ENGINEERING SYSTEM
計測エンジニアリングシステム株式会社
https://kesco.co.jp

1 はじめに

自動テスト環境では、XML-RPCまたはJSON-RPCインタフェースを使用して外部スクリプト経由でRT Boxを制御できます。RT BoxのXML/JSON-RPCインタフェースの詳細については、RT Box User Manual[1]を参照してください。XML-RPCとJSON-RPCは、リモートマシン上で関数を実行するための軽量プロトコルです。RT BoxはXML/JSON-RPCサーバとして機能し、別のコンピュータで実行しているスクリプトから送信された要求を処理します。Pythonなど、多くのスクリプト言語はXML/JSON-RPCを標準でサポートしています。テストの自動化では、XML/JSON-RPCインタフェースをオープンソースの自動化フレームワーク"Robot"と組み合わせて使用できます。

このデモでは、RT BoxのXML/JSON-RPCインタフェースとRobot Frameworkを使用して、RT Boxの基本的な自動テストを設定する方法を示します。

1.1 必要なソフトウェアおよびハードウェア

次のセクションでは、この自動テストのデモを完全に実行するために必要なソフトウェアとハードウェアを示します。PLECSの**ファイル** -> **PLECS設定...** -> **General**で**RPCインターフェイスポート番号**チェックボックスをオンにし、ポートを`1080`に設定して、XML-RPCインタフェースを有効にします。

添付ファイル

このモデルには、PLECSモデル(`.plecs`)と2つのPythonファイル(`.py`)の3つのファイルが含まれています。これらのファイルは、**Help**メニュー -> **PLECS Documentation**の**RT Box Demo Models**セクションにあります。

Pythonのインストール

WindowsおよびMac OSの場合、Python 3.xはhttps://www.python.org/downloads/から新規インストールや更新が行えます。

matplotlibとnumpyのインストール

次のコマンドを入力して`matplotlib`をインストールします:

Python 3.xの場合:

- Windowsのコマンドプロンプトを使用する:

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

  `pip`モジュールが不明であるというエラーメッセージが表示される場合は、`matplotlib`をインストールする前に、まず`pip`モジュールインストールする必要があります。

- Macのターミナルを使用する:

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

`numpy`パッケージのインストールにも同じ方法で行います。

Robot Frameworkのインストール

Robot Frameworkはpipを使ってインストールします。Python 3.xの場合:

- Windowsのコマンドプロンプトを使用する:

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

- Macのターミナルを使用する:

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

詳細なインストール手順については、公式のRobot Framework User Guide[2]のインストール手順の章を参照してください。

ハードウェア

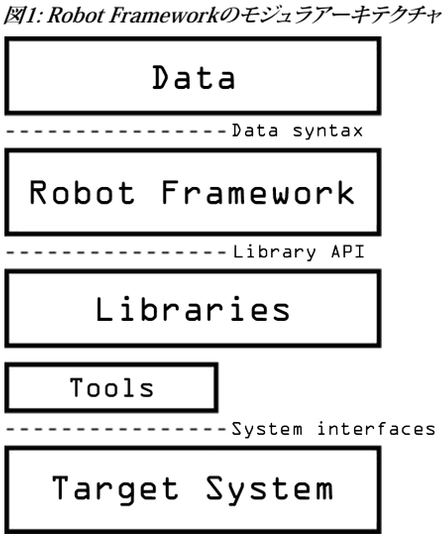
イーサネット経由でホストコンピュータに接続しているRT Box(任意のバージョン)が必要です。さらに、有効なPLECSおよびPLECS Coderのライセンスが必要です。これらの製品の試用ライセンスをリクエストするには、www.plexim.com/trialにアクセスしてください。さらにRT Boxの最新のTarget Support Packageと、Analog Outインタフェースを Analog Inインタフェースに、Digital Outインタフェースを Digital In インタフェースに接続するための、2本の37ピンD-Subケーブルが必要です。

2 Robot Framework

Robot Frameworkはオープンかつ拡張可能であり、事実上あらゆる他のツールと統合して、強力かつ柔軟な自動化ソリューションを作成できます。オープンソースであるということは、Robot Frameworkがライセンス費用なしで無料で使用できることも意味します。Robot Frameworkはキーワード駆動型アプローチを利用します。Pythonで実装されたライブラリによって機能を拡張できます。Robot FrameworkはGitHubで主催しており、詳細なドキュメント、ソース コード、問題追跡ツールを見つけることができます。オペレーティングシステムから独立して実行され、コアフレームワークはPythonで実装されています。

Robot Frameworkには、ユーザ定義のライブラリで拡張できるモジュール型アーキテクチャがあります。データは、以下の例に示す構文を使用してファイル内に定義されます。テストセットを含むファイルは、スイートを作成します。自動化テストを実行する際、フレームワークはまずデータを解析し、次にキーワードを用いてターゲットシステム(PLECS RT Box)と対話します。自動化テストはコマンドラインから開始できます。テスト結果として、HTML形式のレポートとログ、およびXML出力を取得します。これらは、テスト中のシステムの動作の完全な記録を提供します。RT Boxに固有の利用可能なキーワードの概要については、第7章で説明します。

図1: Robot Frameworkのモジュラアーキテクチャ



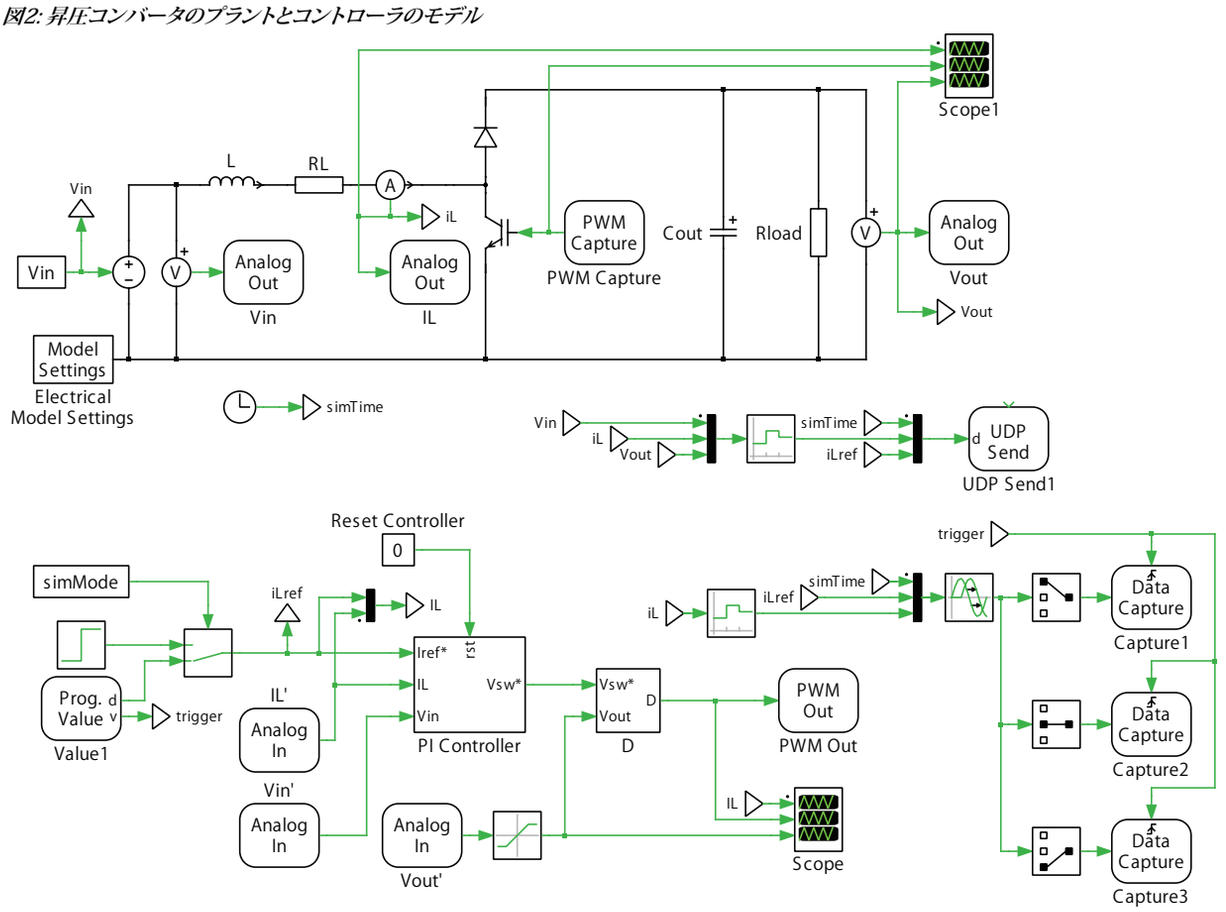
3 モデル

Robot Frameworkを使用したテスト自動化のデモに使用するモデルは、RT Box Demo Modelの**Boost Converter**で説明しいる基本的な昇圧コンバータです。このモデルは、テスト自動化用の特定のブロック(Programmable Valueブロック、Data Captureロック、UDP Sendブロックなど)で拡張されています。

3.1 コントローラ

昇圧コンバータのインダクタ電流は、PI Controllerによって制御されます。コントローラは、`Programmable Value`ブロックから設定ポイントを受け取ります。ブロックがXML/JSON-RPC経由で新しい値を受け取るまで、初期値が出力されます。`Programmable Value`ブロックが新しい値を受信すると、その出力**v**は0から1に変わります。これにより、3つのData Captureブロックのデータ収集をトリガします。リアルタイムシミュレーションが開始されると、UDP Sendブロックは、IPアドレスで指定されたホストコンピュータにネットワーク経由でデータを送信します。

図2: 昇圧コンバータのプラントとコントローラのモデル



4 Robot Frameworkによるテスト自動化

この章では、第3章の昇圧コンバータのキーワード駆動テストについて説明します。Robot Frameworkのデータファイルは、以下に示すようにさまざまなセクションに分かれています:

- Variables: テストデータ内の他の場所で使用できる変数を定義します。
- Settings: 自作のテストライブラリをインポートします。
- Keywords: 既存の低水準キーワードを使用してユーザキーワードを作成します。
- Test Cases: 既存のキーワード(低水準または"Keywords"セクションで定義)からテストケースを作成します。

上記のセクションは、それぞれのヘッダ行によって識別されます(例: `*** Variables ***`)。

4.1 Variables

"Variables"セクションでは、テストデータファイル内の他の場所で使用できる変数を定義できます。このデモでは、RT Box ホスト名、RT Box種類、ホストコンピュータのIPアドレス、UDPブロックのサンプリング時間、UDP経由で送信する信号の数、作成された.csvファイルのヘッダ、およびモデル名が変数として定義されています。

```
*** Variables ***
${RTBOX}=  RTBox.local
${RTBOX_TYPE}=  PLECS RT Box 1
${HOST_PC_IP_ADDRESS}=  255.255.255.255
${RPC_METHOD}=  XML
${UDP_SAMPLE_TIME}=  ${0.001}
${UDP_signal_width}=  ${5}
${UDP_HEADER}=  [time, Vin (V), iL (A), Vout (V), iLref (A)]
${MODEL_NAME}=  boost_converter
```

4.2 Settings

"Settings"セクションは、特定の低水準キーワードを含むテストライブラリをインポートするために使用します。この例では、ライブラリファイルへの物理パスを使用して、RT Box固有のライブラリをインポートする必要があります。パスは常に、現在のテストデータ ファイルが配置されているディレクトリを基準とします。以前に定義された変数`${RPC_METHOD}`が引数としてライブラリに渡されます。最後に、モデル固有のキーワードを含む2番目のライブラリ(`boost_converter.py`)がインポートされます。

```
*** Settings ***
Library  ./PlecsXMLJSONRPC.py  ${RPC_METHOD}
Library  ./boost_converter.py
```

注意 プロジェクト固有の低水準キーワードは、別のライブラリに追加できます。このライブラリは"Settings"セクションでインポートできます。

4.3 Keywords

"Keywords"セクションは、既存の低水準キーワードに基づいて新しいキーワードを定義するために使用します。これは、複数のサブタスクを1つの大きなタスクにグループ化するのに便利です。このデモでは、既存の低水準キーワード`rtBoxSetupServer`、`plecsGenerateCode`、`rtBoxUpload`に基づいて、キーワード`Compile and upload`が生成されます。

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

したがって、新しいキーワード`Compile and upload`を実行すると、まずRT Boxへの XML/JSON-RPC接続が初期化され、次に指定されたモデルからコードが生成されます。これが完了すると、`.elf`ファイルはホスト名で定義された RT Boxに自動的にアップロードされます。

4.4 Test Cases

このセクションで定義されたさまざまなテストケースは常に1つずつ実行し、実行は最上位のテストケースから開始します。通常、いずれかのキーワードが失敗するか、テストケース内のすべてのキーワードを順番に実行すると、現在のテストケースの実行を終了します。このデモでは、次のテストシーケンスを実行します:

```
*** 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  0  2.0
  ${iLref}=  rtBoxGetValue  Capture2  0  2.0
  ${iL}=  rtBoxGetValue  Capture3  0  2.0
  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
```

- まず、モデルを開いてコンパイルし、変数`${RTBOX}`で定義された RT boxにアップロードされます。`plecsGenerateCode`キーワードのオプションの'*optStruct*'引数を使用すると、モデルファイル内のそれぞれのパラメータに異なる値が渡されます。
- その後、`plecsGetCircuitBitmap`キーワードを使用して、RT boxで実行している実際の回路のビットマップが、自動テストの最後に作成されるレポートに記録されます。
- RT boxでのリアルタイムシミュレーションを開始する前に、UDP経由のデータログが有効になります。これは、`rtBoxBackgroundLogging`キーワードを`True`に設定し、正しいパラメータで初期化することによって実現します。
- バックグラウンド ログが有効になると、`rtBoxStart`キーワードを使用してRT Box上のリアルタイムシミュレーションを開始します。
- 2秒間の停止後、"Value1"という名前のProgrammable Valueブロックの出力値を90.0に変更します。これにより、モデル内の3つのData Captureブロックのデータキャプチャがトリガされます。
- キャプチャされたデータは、`rtBoxGetValue`キーワードで読み取られます。
- データをホストコンピュータで受信すると、プロットしてテストレポートに追加します。
- 最後に、UDPバックグラウンドログが無効になり、RT box上のシミュレーションが停止します。

4.5 Robotテストファイルを起動

Robot Frameworkテストを実行する最も簡単な方法は、作業ディレクトリのコンソールを使用することです。コンソールに次の行を入力することで、単一のファイルを実行することができます:

```
robot filename.robot
```

ここで、"*filename*"は実行する.*robot*ファイルの実際の名前を指します。

5 シミュレーション

テストを開始する前に、boost_converter.robot内の変数(${RTBOX}、${RTBOX_TYPE}、${HOST_PC_IP_ADDRESS}、および${RPC_METHOD})を設定に合わせて変更します。適切なテキストエディタでファイルを変更します。変数${RTBOX}は、RT Boxのホスト名を参照します。変数${RTBOX_TYPE}は、RT Boxの製品名を指定します(例: PLECS RT Box 1)。変数${HOST_PC_IP_ADDRESS}は、PLECSとRobot Frameworkを実行しているコンピュータの IP アドレスと一致させる必要があります。変数${RPC_METHOD}は、使用するRPCプロトコル(XMLまたはJSON)を指定します。

```
*** Variables ***
${RTBOX}=  RTBox.local
${RTBOX_TYPE}=  PLECS RT Box 1
${HOST_PC_IP_ADDRESS}=  255.255.255.255
${RPC_METHOD}=  XML
```

Robot Frameworkテストを起動する前に、ホストコンピュータのPLECSを起動しておく必要があります。PLECSが起動すると、コンソールに入力することで自動テストが実行されます。

```
robot boost_converter.robot
```

ターミナルは現在の作業ディレクトリから起動する必要があることに注意してください。テスト中、キーワードの実行が正常に終了するたびに、進行状況がドットで表示されます。テストが終了すると、すべてのテストケースに対してコンソールに"Pass"または"Fail"が表示され、現在の作業ディレクトリにログファイルとHTMLレポートファイルが作成されます。report.htmlファイルは標準のWebブラウザで開くことができます。レポートを参照して、個々のキーワードの結果を確認します。キーワードrtBoxBackgroundLoggingにより、作業ディレクトリに.csvファイルが作成されます。このファイルには、使用したサンプリング時間で等間隔で受信したUDPデータの信号値が含まれています。図4に示すように、3つのData Captureブロックでキャプチャされたデータを含むプロットがreport.htmlファイルに追加されます。

図3: 自動テスト後のコンソールの出力

```
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
```

6 まとめ

このデモモデルは、RT Box上でRobot Frameworkを使用して自動テストを実行する方法を示します。自動テスト中、rtBoxBackgroundLoggingキーワードによって開始したUDPを使用して、関連するすべての信号をホストコンピュータに記録できます。テストの実行後、Robot Frameworkによって包括的なレポートファイルが自動的に作成されます。このテストレポートは、任意のWebブラウザで開くことができます。

図4: RT Boxでキャプチャした現在の参照ステップに対するシステム応答

7 PLECS RT Box用Robot Framework Library

ライブラリは、低水準キーワードを提供することで、Robot Frameworkに実際の自動化およびテスト機能を提供します。フレームワークにはいくつかの標準ライブラリがバンドルされており、専用のターゲットシステム(PLECS RT Box)と対話するための外部ライブラリも別途開発されています。以下に、PLECS RT Boxを操作するために使用できるキーワードの概要を示します。*斜体*で示された引数はすべてオプションです。

XML/JSON-RPCコマンドの概要

キーワード	引数	戻り値
`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`		struct
`rtBoxQuerySimulation`	*`identifier`*	struct
`rtBoxQueryCounter`	*`identifier`*	struct

キーワード	引数	戻り値
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'

現在の作業ディレクトリに対する相対パスで定義されたモデルを開きます。

plecsGenerateCode 'modelName' '*subsystemName*' 'rtboxType' '*optStruct*

*modelName*およびオプションで*subsystemName*によって定義されたモデルのコード生成を開始します。オプションの引数 *rtboxType* は、RT Box 1とは異なるターゲットを定義するために使用できます。この引数の有効なエントリは、PLECS RT Box 1、PLCECS RT Box 2、PLECS RT Box 3です。構造体*optStruct*は、モデルファイルを変更せずに、さまざまなパラメータ値のコードを生成するために使用します。

plecsGetCircuitBitmap '*subsystemName*'

Robot Frameworkによって自動的に生成されるテストレポートにモデルのビットマップを追加します。オプションとして、*subsystemName*を指定してサブシステム内の回路のビットマップを作成することもできます。

rtBoxSetupServer 'rtBoxName'

ホスト名で定義されたRT BoxへのXML/JSON-RPC通信を確立します。

rtBoxStart '*waitForTrigger*'

RT Box上でリアルタイムシミュレーションの実行を開始します。オプションの引数*waitForTrigger*を使用すると、SFP経由で開始トリガを受信するまで、リアルタイムシミュレーションの開始を遅らせることができます。*waitForTrigger*引数は、起動がSFP経由で別のRT Boxと同期している場合にのみ有用です。

rtBoxStop

RT Box上のリアルタイムシミュレーションの実行を停止します。

`rtBoxUpload 'modelName' 'subsystemName'`

RT Boxに実行可能ファイル(ELFファイル)を読み込みます。`.elf`ファイルは、上記で説明した`plecsGenerateCode`キーワードを使用して生成できます。

`rtBoxReboot`

RT Boxを再起動します。再起動中は、RT Boxは応答しなくなります。

`rtBoxGetStatus 'identifier'`

RT Boxの温度、ファン速度、アプリケーションログ、モデルのタイムスタンプを含む構造体を返します。オプションの*identifier*を使用すると、構造体から単一の戻り値を指定できます。*identifier*の有効な値は次のとおりです:

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

`rtBoxGetVersion 'identifier'`

RT Boxのシリアル番号、RT Boxのリビジョンバージョン、CPUシリアル番号、ビルドの日付を含むRT Boxで実行されているファームウェアのビルド番号、ファームウェアバージョン、FPGAバージョン番号、実際のIP アドレス、および RT BoxのMACアドレスを含む構造体を返します。オプションの*identifier*を使用すると、構造体から単一の戻り値を指定できます。*identifier*の有効な値は次のとおりです:

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

`rtBoxGetHostname`

RT Boxの名前を文字列として返します。

`rtBoxQuerySimulation 'identifier'`

RT Boxターゲットサポートパッケージのバージョン番号、RT Boxで実行されているモデルの名前、実際のモデルのタイムスタンプ、およびRT Boxで実行されているモデルの離散化ステップサイズを含む構造体を返します。オプションの*identifier*を使用すると、構造体から単一の戻り値を指定できます。*identifier*の有効な値は次のとおりです:

```
'applicationVersion'、'modelName'、'modelTimeStamp'、'sampleTime'
```

`rtBoxQueryCounter 'identifier'`

RT Boxで実行されているモデルの最大サイクル時間、モデルのタイムスタンプ、および現在のサイクル時間を含む構造体を返します。オプションの*identifier*を使用すると、構造体から単一の戻り値を指定できます。RT Box 1の*identifier*の有効な値は次のとおりです:

```
'maxCycleTime'、'modelTimeStamp'、'runningCycleTime'
```

RT Box 2および3では、次の値が許可されます。

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

インデックス番号(1、2、3)は、RT Box 2および3のそれぞれのコアを表します。

rtBoxResetCounter

RT Boxの**maximum cycle time**値をリセットします。

rtBoxGetLEDs

RT Box 1の前面パネルにある4つのLEDのステータスを含む構造体を返します。各LEDの戻り値は、LEDが点灯している場合は'1'、LED が消灯している場合は' 'になります。このキーワードはRT Box 1でのみ使用できます。

rtBoxSetValue 'path' 'value'

*path*で示されるProgrammable Valueブロックの出力を設定します。パラメータの*path*は、コード生成サブシステムを基準としたブロックのパスです。パラメータ*value*は、要素数が出力信号の幅に対応するXML/JSON-RPC配列である必要があります。

rtBoxGetValue 'path' '*triggerCount*'

*path*で示されるData Captureブロックから最後に埋められたデータバッファを返します。指定されている場合、キーワードは、オプションの引数*triggerCount*で指定された回数だけサンプルバッファが埋められるまで待機します。

rtBoxCaptureTriggerCount 'path'

*path*で示されるData Captureブロックのサンプルバッファが何回埋められたかを返します。

rtBoxWaitForTrigger 'path' 'timeout'

*path*で指定されたData Captureブロックのサンプルバッファが埋められるまで待機します。待機時間がパラメータ*timeout*で指定された間隔を超えると例外が発生し、catchされない限り実行が停止します。

rtBoxIsRunning

RT Box上でリアルタイムシミュレーションが実行されている場合は`True`を返します。

rtBoxGetDataCaptureBlocks

現在のモデル内のすべてのData Captureブロックのリストをパスとともに返します。

rtBoxGetProgrammableValueBlocks

現在のモデル内のすべてのProgrammable Valueブロックのリストをパスとともに返します。

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

引数*enable*が`True`に設定されている場合、Robot Frameworkを使用した自動テスト中にバックグラウンドログ記録が有効になります。バックグラウンド ログ記録キーワードは、ポート**52345**でリッスンするUDPソケットを設定するバックグラウンドスレッドを開始します。バックグラウンドログ記録を機能させるには、モデルファイルに少なくとも1つのUDPブロックが必要です。さらに、UDP Sendブロックで設定されたリモートIPアドレスは、ホストPCのIPアドレスと一致する必要があります。バックグラウンドログ記録は常にポート番号**52345**を使用します。引数 *signalWidth*は、PLECSモデル内のUDP Sendブロックに接続されている信号の数と一致する必要があります。引数*sampleTime*についても同様です。許容される最速のサンプル時間は1ミリ秒です。キーワードを使用して引数*enable*を`False`に設定し、バックグラウンドログ記録を無効にする場合、他のすべての引数を指定する必要はありません。

`rtBoxBackgroundloggingEnabled`

バックグラウンドログが有効になっている場合は`True`を返します。有効になっていない場合は`False`を返します。

8 参考文献

[1] *RT Box User Manual*, Plexim GmbH, Online: https://www.plexim.com/sites/default/files/rtboxmanual.pdf
日本語版: https://adv-auto.co.jp/products/plexim/manual.html

[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]

改訂履歴:

RT Box TSP 2.0.4	初版
RT Box TSP 3.1.5	JSON-RPCとPLECS RT Box 2および3のサポートを追加

Pleximへの連絡方法:

☎ +41 44 533 51 00 Phone

+41 44 533 51 01 Fax

✉ Plexim GmbH Mail
Technoparkstrasse 1
8005 Zurich
Switzerland

@ info@plexim.com Email

http://www.plexim.com Web

KESCO 計測エンジニアリングシステムへの連絡方法:

☎ +81 3 6273 7505 Phone

+81 3 6285 0250 Fax

✉ Keisoku Engineering Systm CO.,LTD. Mail
1-9-5 Uchikanda, Chiyoda-ku
Tokyo, 101-0047
Japan

https://kesco.co.jp Web

RT Box Demo Model