Test Automation

Python Modules Reference

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

Contents

About this Reference

Content

This reference provides detailed information on the Python commands of the Test Automation Python Modules.

It is assumed that you know the Test Automation Python Modules Guide 🕮.

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.
🕮	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

Some software products use the following 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.

Safety Precautions

Introduction

To avoid risk of injury and/or property damage, read and ensure compliance with the safety precautions given.

General Warning

Danger potential

Using dSPACE software can be dangerous. You must observe the following safety instructions and the relevant instructions in the user documentation.

Improper or negligent use can result in serious personal injury and/or property damage

Using the dSPACE software can have a direct effect on technical systems (electrical, hydraulic, mechanical) connected to it.

The risk of property damage or personal injury also exists when the dSPACE software is controlled via an automation interface. The dSPACE software is then part of an overall system and may not be visible to the end user. It nevertheless produces a direct effect on the technical system via the controlling application that uses the automation interface.

- Only persons who are qualified to use dSPACE software, and who have been informed of the above dangers and possible consequences, are permitted to use this software.
- All applications where malfunctions or operating errors involve the danger of injury or death must be examined for potential hazards by the user, who must if necessary take additional measures for protection (for example, an emergency off switch).

Liability

It is your responsibility to adhere to instructions and warnings. Any unskilled operation or other improper use of this product in violation of the respective safety instructions, warnings, or other instructions contained in the user documentation constitutes contributory negligence, which may lead to a limitation of liability by dSPACE GmbH, its representatives, agents and regional dSPACE companies, to the point of total exclusion, as the case may be. Any exclusion or limitation of liability according to other applicable regulations, individual agreements, and applicable general terms and conditions remain unaffected.

Data loss during operating system shutdown

The shutdown procedure of Microsoft Windows operating systems causes some required processes to be aborted although they are still being used by dSPACE software. To avoid data loss, the dSPACE software must be terminated manually before a PC shutdown is performed.

Overview of the Python Modules for Test Automation

Overview of the Test Automation Python Modules

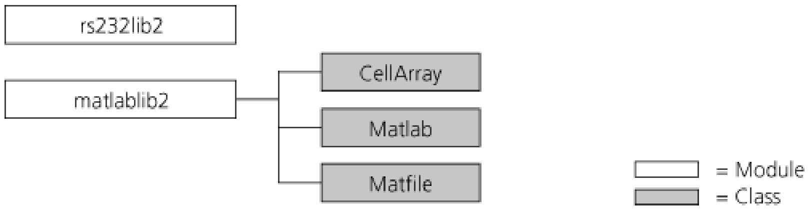
Introduction

The Test Automation Python Modules include specific Python modules used for automating tests.

These modules allow the remote control of MATLAB and laboratory devices connected via serial interface.

Overview

The illustration gives an overview of the modules and classes provided by the Test Automation Python Modules.



The table provides descriptions of the modules:

Module	Description
rs232lib2	To communicate via the serial interface. See Acquiring Data from External Devices (rs232lib2) on page 11.
matlablib2	To exchange data between MATLAB and the Python Interpreter, invoke MATLAB functions or to access files in the MATLAB file format. See Interfacing MATLAB (matlablib2) on page 23.

Packaging and licences

For more information on packages and licences, refer to Introduction to the Python Modules for Test Automation (Test Automation Python Modules Guide 🕮).

Quick reference

For the Test Automation Python Modules, an introduction is available. The object information is summarized in a set of compact tables, each of which provides a quick overview of the available objects, object dependencies, attributes and methods. For a printable version of the quick reference, refer to dSPACE Help.

Multiple identifiers

In the Test Automation Python Modules, some method and class identifiers are multiply used. For easier access to the related documentation in dSPACE Help, see Multiple Use of Identifiers Within the Test Automation Python Modules on page 61.

Related topics

References

Acquiring Data from External Devices (rs232lib2)

Where to go from here

Information in this section

Overview of the rs232lib2 Module

Introduction

The rs232lib2 module provides the functions for communication via the serial interface. The major functions are:

- Opening and closing the connection to the serial interface
- Configuring baudrate, parity, data and stop bits
- Setting input and output buffers
- Receiving data from and sending data to the serial interface

For example, you can use the rs232lib2 functions to access external diagnosis devices or to control laboratory devices remotely.

For further information on the rs232lib2, refer to Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮).

Note

To access a serial port with the rs232lib2, it is required that the serial device driver of the operating system (serial.sys) is started.

Functions

The rs232lib2 module provides following functions:

Purpose	Refer to
Initialization functions	
To close the open connection and delete the created handle.	Close on page 12
To open the serial port and create a handle.	Open on page 14
Configuration functions	
To change the buffer size.	SetBuffers on page 18
To configure an open connection.	SetConfig on page 17
To change the read timeout.	SetReadTimeout on page 19
Input/output functions	
To read a value or several values.	Read on page 15
To write a byte.	Write on page 20
To write a string.	WriteString on page 21
Information functions	
To get the number of bytes in the read buffer.	GetNumInBytes on page 13

Close

Module

rs232lib2

Syntax

```
rs232lib2.Close(hComRS232)
```

Example

```
import rs232lib2
h = rs232lib2.Open("COM1")
...
rs232lib2.Close(h)
```

Purpose

To close the open connection and delete the created handle.

Parameters

The function uses the following parameter:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection. This handle becomes invalid via this function. The Python object itself is not deleted. You have to delete it manually afterwards by assigning `None` to the object.

Return value

–

Related methods

Open on page 14

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

References

Acquiring Data from External Devices (rs232lib2).. 11

GetNumInBytes

Module

rs232lib2

Syntax

```
val = rs232lib2.GetNumInBytes(hComRS232)
```

Example

```
n = rs232lib2.GetNumInBytes(h)
```

Purpose

To get the number of bytes in the read buffer.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection

Return value

The function returns a value of the following type:

Type	Description
Int	Number of bytes in read buffer

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

Open

Module

rs232lib2

Syntax

```
hComRS232 = rs232lib2.Open(pcPort)
```

Example

```
import rs232lib2
h = rs232lib2.Open("COM1")
```

Purpose

To open the serial port and create a handle.

Description

The read timeout is set to a default value of 10 seconds. The input and output buffers have a default size of 1024 bytes.

Parameters

The function uses the following parameters:

Parameter	Type	Description	Unicode Support
pcPort	String	Name of the PC port. Up to 4 PC ports are supported, which means that the parameter pcPort may be "COM1" ... "COM4".	Yes

Return value

The function returns a value of the following type:

Type	Description
Int	A handle to the opened PC port or the established connection. This handle is to be used for each subsequent configuration, read/write and info function call.

Related methods

Close on page 12

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 📖)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 📖)

Read

Module

rs232lib2

Syntax

```
str = rs232lib2.Read(hComRS232, BytesToRead)
```

Example

```
import rs232lib2
vals = rs232lib2.Read(h, 5)
```

Purpose

To read a value or several values.

Description

Byte-oriented reading of data from input buffer of serial PC port.

Note

Be aware of the following cases of Timeout behavior depending on the state of data in input buffer. After a call of the Read function, one of the following cases occur:

- There are at least as many bytes in input buffer as specified for reading. The function is finished by returning the read bytes.
- One byte is to be read and there is no data in the input buffer. The function is finished as soon as a byte comes into the input buffer within the specified time out. If the time out is reached without a new byte coming into the input buffer, the read function is aborted with an exception.
- More than one byte is to be read. No values are stored in the input buffer when calling the function. The function is finished with an exception as soon as a byte is transferred into the read buffer or the time out is reached.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection
BytesToRead	Int	Number of bytes to read. As a default, this parameter is set to 1 byte.

Return value

The function returns a value of the following type:

Return Value	Type	Description	Unicode Support
str	String	String read from the PC port	No

Related methods

Write on page 20

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

SetConfig

Module

rs232lib2

Syntax

```
rs232lib2.SetConfig(hComRS232, BaudRate, BitNumber, Parity,
                             StopBits)
```

Example

```
import rs232lib2
rs232lib2.SetConfig(h, 4800, 8, "NO", 1)
```

Purpose

To configure an open connection.

Parameters

The function uses the following parameters:

Parameter	Type	Description	Unicode Support
hComRS232	Int	A handle to the opened PC port or the established connection	-
BaudRate	Int	Baudrate at which the communication port operates (valid values are: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200)	-
BitNumber	Int	Number of data bits to be used (valid values are: 5, 6, 7, 8).	-
Parity	String	Parity scheme to be used (valid values are: 'NO', 'ODD', 'EVEN', 'MARK', or 'SPACE')	No
StopBits	Int/Float	Number of stop bits to be used (valid values are: 1, 1.5, 2)	-

Note

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

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 📖)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 📖)

SetBuffers

Module

rs232lib2

Syntax

```
rs232lib2.SetBuffers(hComRS232, InBufferSize, OutBufferSize)
```

Example

```
import rs232lib2
rs232lib2.SetBuffers(h, 2048, 2048)
```

Purpose

To change the buffer size.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection
InBufferSize	Int	Size of the read buffer in bytes, usually 1024 or 2048 bytes
OutBufferSize	Int	Size of the write buffer in bytes, usually 1024 or 2048 bytes

Note

Changing the buffer size deletes the data in the buffer.

Return value

–

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

SetReadTimeout

Module

rs232lib2

Syntax

```
rs232lib2.SetReadTimeout(hComRS232, ReadTimeout)
```

Example

```
import rs232lib2
rs232lib2.SetReadTimeout(h, 8000)
```

Purpose

To change the read timeout

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection
ReadTimeout	Int	Timeout value for reading in milliseconds

Return value

–

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

Write

Module

rs232lib2

Syntax

```
rs232lib2.Write(hComRS232, ByteToWrite)
```

Example

Write the ASCII value 65, that is, "A" to the open PC port connection. For conversion of characters to the ASCII value, the Python built-in function ord can be used.

```
import rs232lib2
rs232lib2.Write(h, 65)
rs232lib2.Write(h, ord('A'))
```

Purpose

To write a byte.

Parameters

The function uses the following parameters:

Parameter	Type	Description
hComRS232	Int	A handle to the opened PC port or the established connection
ByteToWrite	Int	ASCII value of the byte to be written

Return value

–

Related methods

Read on page 15

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 🕮)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 🕮)

WriteString

Module

rs232lib2

Syntax

```
rs232lib2.WriteString(hComRS232, StringToWrite)
```

Example

Writes a string into the output buffer:

```
rs232lib2.WriteString(h,'Test')
```

Purpose

To write a string.

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	Unicode Support
hComRS232	Int	A handle to the opened PC port or the established connection	-
StringToWrite	String	String to be written to the opened PC port	No

Return value

–

Related topics

Basics

Acquiring Data from External Devices (Test Automation Python Modules Guide 📖)

Examples

Example of Accessing the Serial Interface (Test Automation Python Modules Guide 📖)

Interfacing MATLAB (matlablib2)

Where to go from here

Information in this section

Overview of the matlablib2 Module

Where to go from here

Information in this section

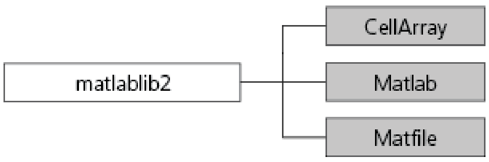
Basic Information on the matlablib2 Module

Introduction

The matlablib2 module provides access to MATLAB. Using the class Matlab, you can exchange data between MATLAB and the Python Interpreter or invoke MATLAB functions (use existing m-scripts, for example). matlablib2 also provides access to the MATLAB file format (MAT files) via the class Matfile. For information on how to use matlablib2, refer to Working with MATLAB (Test Automation Python Modules Guide 📖).

Classes

The matlablib2 module defines the following classes:



	Purpose	Refer to
(CaptureData class)[1)]		
CellArray class		
	To provide a Python representation of the MATLAB cell array type.	CellArray on page 29
Matfile class		
	To create an instance of the class Matfile.	Matfile on page 35
Matlab class		
	To create an instance of the class Matlab.	Matlab on page 46

1) Discontinued as of dSPACE Release 2021-A.

Exception

matlablibError

Constants

The following table shows the constants and their description:

Constant	Description
CellArrayType	The type of a CellArray instance

Examples

For examples, refer to Examples of Using matlablib2 on page 60.

MATLAB Type Conversion

MATLAB type conversion

MATLAB's array classes are mapped to the Python types int, long, float, string, complex, list (of lists), dictionary and CellArray instances when reading from a MAT file or the MATLAB Workspace. Vice versa these Python types are mapped to MATLAB array classes when writing to a MAT file or the MATLAB Workspace.

The following table shows some examples of type conversions that apply to both directions:

MATLAB Array Class	Python Type
1x1 int32 array	int
1x1 double array	float
1x1 double array (complex)	complex
1xN char array	string
1x1 struct array	dictionary
1xN int32 array	list of int
1xN double array	list of float
1xN double array (complex)	list of complex
MxN char array	list of string
1xN struct array	list of dictionary
MxN int32 array	list of list of int
MxN double array	list of list of float
MxN double array (complex)	list of list of complex
MxNxO char array	list of list of string
MxN struct array	list of list of dictionary
MxN cell array	CellArray instance (MxN)

Lists of dimension > 2 are converted to the appropriate multi- dimensional MATLAB array class.

The following table shows the exact type conversion rules for MATLAB basic types while reading from a MAT file or the MATLAB Workspace:

From MATLAB Array Class	To Python Type	Description
0x0 double array	ListType	Empty list
1x1 double array	FloatType	-
1xN double array	ListType	List of values of type FloatType
MxN double array	ListType	List of values of type ListType, each of the inner lists containing values of type FloatType
MxNx... double array	ListType	List of lists of lists of … type FloatType
1xN char array	StringType	-
MxN char array	ListType	List of values of type StringType
MxNx... char array	ListType	List of lists of … StringType
1x1 struct array	DictionaryType	Keys of type StringType, values of appropriate types
1xN struct array	ListType	List of values of type DictionaryType, keys of type StringType, values of appropriate types
MxN struct array	ListType	List of lists of values of type DictionaryType, keys of type StringType, values of appropriate types
MxNx... struct array	ListType	List of lists of lists of … type DictionaryType, keys of type StringType, values of appropriate types
0x0 cell array	CellArrayType	Empty cell array
1x1 cell array	CellArrayType	Each cell contains a value of appropriate Python type
1xN cell array	CellArrayType	Each cell contains a value of appropriate Python type
MxN cell array	CellArrayType	Each cell contains a value of appropriate Python type
MxNx... cell array	CellArrayType	Each cell contains a value of appropriate Python type
1x1 double array (complex)	ComplexType	-
1xN double array (complex)	ListType	List of values of type ComplexType
MxN double array (complex)	ListType	List of values of type ListType, each of the inner lists containing values of type ComplexType
MxNx... double array (complex)	ListType	List of lists of lists of … type ComplexType
1x1 single array	FloatType	-
1xN single array	see row: 1xN double array	-
MxN single array	see row: MxN double array	-
MxNx... single array	see row: MxNx... double array	-
1x1 int8 array	IntType	-
1xN int8 array	ListType	List of values of type IntType
MxN int8 array	ListType	List of values of type ListType, each of the inner lists containing values of type IntType

From MATLAB Array Class	To Python Type	Description
MxNx... int8 array	ListType	List of lists of lists of … type IntType
1x1 int16 array	IntType	-
1xN int16 array	see row: 1xN int8 array	-
MxN int16 array	see row: MxN int8 array	-
MxNx... int16 array	see row: MxNx... int8 array	-
1x1 int32 array	IntType	-
1xN int32 array	see row: 1xN int8 array	-
MxN int32 array	see row: MxN int8 array	-
MxNx... int32 array	see row: MxNx... int8 array	-
1x1 uint8 array	IntType	-
1xN uint8 array	see row: 1xN int8 array	-
MxN uint8 array	see row: MxN int8 array	-
MxNx... uint8 array	see row: MxNx... int8 array	-
1x1 uint16 array	IntType	-
1xN uint16 array	see row: 1xN int8 array	-
MxN uint16 array	see row: MxN int8 array	-
MxNx... uint16 array	see row: MxNx... int8 array	-
1x1 uint32 array	LongType	-
1xN uint32 array	ListType	List of values of type LongType
MxN uint32 array	ListType	List of values of type ListType, each of the inner lists containing values of type LongType
MxNx... uint32 array	ListType	List of lists of lists of … type LongType

The following table shows the exact type conversion rules for Python basic types while writing to a MAT file or the MATLAB Workspace:

From Python Type	To MATLAB Array Class	Description
FloatType	1x1 double array	-
IntType	1x1 int32 array	-
LongType	1x1 int32 array	If long value <= 2^31-1
ComplexType	1x1 double array (complex)	-
StringType	1xN char array	-
UnicodeType	1xN char array	-
DictionaryType	1x1 struct array	All keys of the dictionary must be of type StringType
ListType	0x0 double array 1xN array of the class double, int32, complex, char or struct MxNx... array of the class double, int32, complex, char or struct	If list is empty If list contains values of the basic Python types float, int, long, complex, string, unicode object, or dictionary, resp. Multidimensional MATLAB array of appropriate type

From Python Type	To MATLAB Array Class	Description
CellArrayType	0x0 cell array MxNx... cell array	If Python CellArray is empty else

The following examples show how to specifiy row or column vectors, matrices and cell arrays in Python:

1x1 vector (a plain value)	1.0 = [1.0] = [[1.0]]
1x3 row vector	[1.0, 2.0, 3.0] = [[1.0, 2.0, 3.0]]
3x1 column vector	[[1.0], [2.0], [3.0]]
2x3 matrix	[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]
1x1 integer vector	1 = [1] = [[1]]
1x3 integer row vector	[1, 2, 3] = [[1, 2, 3]]
3x1 integer column vector	[[1], [2], [3]]
2x3 integer matrix	[[1, 2, 3], [4, 5, 6]]

2x2 cell array	c1 = CellArray(2,2)
	c1.SetItem(0,0, [1.0,2.0])
	c1.SetItem(0,1, "dSPACE")
	c1.SetItem(1,0, [1])
	c2 = CellArray(1,2)
	c2.SetItem(0,0, [1])
	c2.SetItem(0,1, [2.0,4.0])
	c1.SetItem(1,1, c2)

In MATLAB all arrays are at least two-dimensional. As one can see from the examples and Interfacing MATLAB (matlablib2), it is possible to put plain values to MATLAB via PutArray() as an abbreviation for specifying a list containing a list that contains a single value (see example 1x1 vector above). The same approach applies to row vectors: one can simply specify a Python list containing the values as an abbreviation for a list containing a single list containing the column values (see example 1x3 row vector above).

CellArray

Where to go from here

Information in this section

CellArray Class Description

Module

matlablib2

Syntax

Two possibilities:

```
OBJ = matlablib2.CellArray(ListOfDimensionSizes)
```

or

```
OBJ = matlablib2.CellArray([N1 [,N2 [,N3]]])
```

Example

```
import matlablib2
c1 = matlablib2.CellArray(2,2)
```

Purpose

To provide a Python representation of the MATLAB cell array type.

Description

The Python representation of a MATLAB cell array is an instance of the class CellArray, defined in module matlablib2. The constructor of CellArray may be used in one of two ways:

- The first form takes a list as argument, containing the number of elements for each dimension of the resulting CellArray instance. The length of the list determines the dimension of the CellArray. Due to limitations in MATLAB, the dimension is at least 2. Therefore, if the constructor argument is [], the resulting CellArray has size 0x0, if the argument is [N1], the resulting CellArray has size N1x0. if the argument is [N1, N2], the resulting CellArray has size N1xN2, and so on.

- As a abbreviation, the second form takes up to three arguments of type int. If no argument is given, an instance of an empty cell array is created (dimension 2, 0x0). If only N1 is given, the resulting CellArray instance has dimension 2 (N1x0). If in addition to N1 an argument N2 is given, the resulting CellArray instance also has dimension 2 (N1xN2). If (in addition to N1 and N2) N3 is given the resulting CellArray instance has dimension 3 (N1xN2xN3).

Parameters

The following parameters are used:

Parameter	Type	Description
ListOfDimensionSizes	List	To set a list containing the size of each dimension of the CellArray instance to be created.
N1	Int	To set the size of the first dimension.
N2	Int	To set the size of the second dimension.
N3	Int	To set the size of the third dimension.

Return value

The following object is returned:

Return Value	Type	Description
OBJ	CellArray	The created CellArray object

Exception

The following exception may be raised:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Attributes

–

Methods

The class CellArray provides the following methods:

Method	Purpose
GetDimensions	To get the size of each dimension. See GetDimensions on page 31.
GetItem	To get a cell array item. See GetItem on page 31.
GetNumberOfDimensions	To get the number of dimensions. See GetNumberOfDimensions on page 32.
SetItem	To set a cell array item. See SetItem on page 33.

GetDimensions

Class	CellArray
Syntax	`[Result = ] OBJ.GetDimensions()`
Purpose	To get the size of each dimension.
Description	This method returns a list containing the sizes for each dimension.
Parameters	–

Return value

The method returns a value of the following type:

Type	Description
List	A list containing the size of each dimension. The length of the list is at least 2.

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

GetNumberOfDimensions on page 32

GetItem

Class	CellArray

Syntax	`[Result = ] OBJ.GetItem(IndexList)`
	`[Result = ] OBJ.GetItem(I1, I2 [,I3])`

Purpose	To get a cell array item.

Description

The content of the cell at the indices specified by IndexList is returned.

As an abbreviation, the second form of GetItem takes 2 or 3 arguments of type int.

Parameters

The method uses the following parameters:

Parameter	Type	Description
IndexList	List of Int	The list of indices specifying the cell to return. This list must contain values of type Int. Since a CellArray is at least 2-dimensional, the length of the list must be at least 2.
I1	Int	First dimension index of cell to be returned
I2	Int	Second dimension index of cell to be returned
I3	Int	Third dimension index of cell to be returned

Return value

The method returns a value of the following type:

Type	Description
Various	The Python representation of the cell array. See Interfacing MATLAB (matlablib2) on page 23 for a detailed description of valid Python value types.

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

SetItem on page 33

GetNumberOfDimensions

Class

CellArray

Syntax

```
[Result = ] OBJ.GetNumberOfDimensions()
```

Purpose

To get the number of dimensions.

Description

This method returns the number of dimensions of the CellArray instance. Due to MATLAB's representation of arrays, this number is at least 2.

Parameters

–

Return value

The method returns a value of the following type:

Type	Description
Int	The number of dimensions

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

GetDimensions on page 31

SetItem

Class

CellArray

Syntax

```
OBJ.SetItem(IndexList, Value)
```

```
OBJ.SetItem(I1, I2 [,I3], Value)
```

Purpose

To set a cell array item.

Description

The content of the cell at the indices specified by IndexList is set.

As an abbreviation, the second form of SetItem takes 2 or 3 index arguments of type int (and the value to set).

Parameters

The method uses the following parameters:

Parameter	Type	Description
IndexList	List of Int	The list of indices specifying the cell to be set. This list must contain values of type int. Since a MATLAB cell array is at least 2-dimensional, the length of the list must be at least 2.
I1	Int	First dimension index of cell to be set
I2	Int	Second dimension index of cell to be set
I3	Int	Third dimension index of cell to be set
Value	Various	Python representation of the cell content to be set

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

–

Matfile

Where to go from here

Information in this section

Matfile Class Description

Module

matlablib2

Syntax

```
OBJ = matlablib2.Matfile()
```

Example

```
import matlablib2
MyMatfile = matlablib2.Matfile()
```

Purpose

To create an instance of the class Matfile.

Description

An instance of the class Matfile can be used to access MAT files. While reading a MAT file, MATLAB's array classes are converted to Python types. While writing a MAT file, certain Python types are converted to MATLAB arrays.

See Interfacing MATLAB (matlablib2) on page 23 for a detailed list of supported MATLAB array classes and their conversion to an appropriate Python representation.

Parameters –

Return value

The following object is returned:

Return Value	Type	Description
OBJ	Matfile	The created Matfile object

Exception

The following exception may be raised:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Attributes –

Methods

The class Matfile provides the following methods:

Method	Purpose
Close	To close a MAT file. See Close on page 37.
DeleteArray	To delete an array from a MAT file. See DeleteArray on page 37.
GetArray	To get an array from a MAT file. See GetArray on page 38.
GetDir	To get the directory of a MAT file. See GetDir on page 39.
Load	To load the whole MAT file and assign all contained arrays to variables in the global namespace with the same names. See Load on page 40.
Open	To open a MAT file. See Open on page 41.
PutArray	To put an array represented as Python object to a MAT file. See PutArray on page 42.
PutArrayAsGlobal	To put an array represented as Python object to a MAT file as global. See PutArrayAsGlobal on page 43.
Whos	To list variables in the MAT file in long form. See Whos on page 44.

Close

Class	Matfile
Syntax	`OBJ.Close()`
Purpose	To close a MAT file.
Description	This method closes the previously opened MAT file.
Parameters	–
Return value	–
Exception	This method may raise the following exception:

Exception	**Description**
matlablibError	The file could not be closed.

Related methods	Load on page 40, Open on page 41

DeleteArray

Class	Matfile
Syntax	`OBJ.DeleteArray(NameOfArray)`
Purpose	To delete an array from a MAT file.
Description	This method deletes the array with the specified name from the MAT file.

Parameters

The method uses the following parameter:

Parameter	Type	Description
NameOfArray	String Unicode	The name of the array to be deleted.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

GetArray on page 38

GetArray

Class

Matfile

Syntax

```
[Result =] OBJ.GetArray(NameOfArray)
```

Purpose

To get an array from a MAT file.

Description

This method gets the array with the specified name from the MAT file and returns its Python representation.

Parameters

The method uses the following parameters:

Parameter	Type	Description
NameOfArray	String Unicode	The name of the array to be retrieved.

Return value

The method returns a value of the following type:

Type	Description
Various	The Python representation of the MATLAB-array. See Interfacing MATLAB (matlablib2) on page 23 for a detailed description of valid Python value types.

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

DeleteArray on page 37

GetDir

Class

Matfile

Syntax

```
[Result =] OBJ.GetDir()
```

Purpose

To get the directory of a MAT file.

Description

This method gets the directory of a MAT file and returns a list of name strings of all of the arrays contained in the file.

Parameters

–

Return value

The method returns a value of the following type:

Type	Description
List	A list of array name strings

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Load

Class

Matfile

Syntax

```
OBJ.Load()
```

Purpose

To load the whole MAT file and assign all contained arrays to variables in the global namespace with the same names.

Description

This method loads each array contained in the MAT file. Each array is assigned to a variable with the same name in the top-level script environment (the global namespace). This method is similar to MATLAB's `load` command.

Note

Existing variables with the same name are overwritten without notice. Be sure not to overwrite variables that are needed for the currently running script.

Parameters

–

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

Close on page 37,

Open on page 41

Open

Class	Matfile
Syntax	`OBJ.Open(FileName, Mode [, ConvertToDouble=0])`
Purpose	To open a MAT file.
Description	This method allows you to open MAT files for reading and writing.

Parameters

The method uses the following parameters:

Parameter	Type	Description
FileName	String Unicode	The name of the file to open.
Mode	String Unicode	The file opening mode.
ConvertToDouble	Boolean	If this optional parameter is set to 1: ▪ All subsequent calls to PutArray convert Python integer types like int and long to the MATLAB type double. ▪ All subsequent calls to GetArray convert MATLAB integer types to Python float. The default for this parameter is 0 (false).

The following table shows all possible values for the `Mode` parameter:

Mode String	Description
`"r"`	Opens the file for reading only; determines the current version of the MAT file by inspecting the files and preserves the current version.
`"u"`	Opens the file for update, both reading and writing, but does not create the file if the file does not exist (equivalent to the `"r+"`-mode of fopen); determines the current version of the MAT file by inspecting the files and preserves the current version.
`"w"`	Opens the file for writing only; deletes previous contents, if any.
`"w4"`	Creates a MAT file that is compatible with MATLAB version 4 and earlier.
`"wL"`	Opens the file for writing character data using the default character set for your system. The resulting MAT file can be read with MATLAB version 6 or 6.5. If you do not use the wL mode switch, MATLAB writes character data to the MAT file using Unicode encoding by default.
`"wz"`	Opens the file for writing compressed data. The same compression ratio is applied than by saving workspace variables to a MAT file.

Mode String	Description
"w7.3"	Creates a MAT file in an HDF5-based data format. This file format can store objects that require more than 2 GB. This is the default file format that can be read with MATLAB version 7.3 and later.

Return value –

Exception

This method may raise the following exception:

Exception	Description
matlablibError	The file could not be opened.

Related methods

Close on page 37,

Load on page 40

PutArray

Class Matfile

Syntax

```
OBJ.PutArray(NameOfArray, Value)
```

Purpose To put an array represented as Python object to a MAT file.

Description This method puts the array with the specified name and value, represented as Python object, to the MAT file.

Parameters

The method uses the following parameters:

Parameter	Type	Description
NameOfArray	String Unicode	The name of the array to be written.
Value		The Python representation of the MATLAB-array. See Interfacing MATLAB (matlablib2) on page 23 for a detailed description of valid Python value types.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

PutArrayAsGlobal on page 43

PutArrayAsGlobal

Class

Matfile

Syntax

```
OBJ.PutArrayAsGlobal(NameOfArray, Value)
```

Purpose

To put an array represented as Python object to a MAT file as global.

Description

This method puts the array with the specified name and value, represented as Python object, to the MAT file. It is similar to PutArray(), except the array is loaded by MATLAB into the global workspace and a reference to it is set in the local workspace. If you write to a MATLAB 4 format MAT file, `PutArrayAsGlobal` does not load it as global, and acts the same as `PutArray`.

Parameters

The method uses the following parameters:

Parameter	Type	Description
NameOfArray	String Unicode	The name of the array to be written
Value	Various	The Python representation of the MATLAB-array. See Interfacing MATLAB (matlablib2) on page 23 for a detailed description of valid Python value types.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Related methods

PutArray on page 42

Whos

Class

Matfile

Syntax

```
[Result =] OBJ.Whos()
```

Purpose

To list variables in the MAT file in long form.

Description

This method lists all the variables in the MAT file, together with information about their size, bytes, class, etc., and returns this information as a string. The method's behavior is very similar to MATLAB's command `whos`, except for the summary line. Unfortunately, due to lacks of the MATLAB MAT file API, the size of struct arrays can only be determinated as a rough approximation of the real value.

Note

For variables of any class except struct arrays, only the header information is loaded into the memory; struct arrays are loaded completely, and therefore memory consumption may be vast depending on the size of the struct arrays contained in the file.

Parameters

–

Return value

The method returns a value of the following type:

Type	Description
String	A string containing all variable names, dimensions, size and class information

Exception

This method may raise the following exception:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Matlab

Where to go from here

Information in this section

Matlab Class Description

Module

matlablib2

Syntax

```
OBJ = matlablib2.Matlab()
```

Example

```
import matlablib2
MyMatlab = matlablib2.Matlab()
```

Purpose

To create an instance of the class Matlab.

Description

An instance of the class Matlab can be used to transport MATLAB arrays from MATLAB to Python and vice versa, and to execute MATLAB commands. Outputs of the MATLAB workspace can be read by Python applications. The MATLAB Command Window can be minimized and restored.

Parameters

–

Return value

The following object is returned:

Return Value	Type	Description
OBJ	Matlab	The created Matlab object

Exception

The following exception may be raised:

Exception	Description
matlablibError	On any error of this method, matlablibError is returned.

Attributes

The following attributes are part of the class:

Attribute	Type	Description
ExecutablePath	String	To get the path to the executable of the connected MATLAB instance. See ExecutablePath on page 50.
ConnectedMATLABInstallations	List of tuples	To get a list of the connected MATLAB installations containing the installation paths and whether they are configured as the preferred MATLAB instance. See ConnectedMATLABInstallations on page 49.
IsMUMatlabOpen	Tuple (Int, String, String)	To get the flag of whether the connected MATLAB instance is enabled for multiple use. See IsMUMatlabOpen on page 54.
ProcessArchitecture	Int	To get the process architecture (32-bit or 64-bit) of the connected MATLAB instance. See ProcessArchitecture on page 56.
ProcessID	Int	To get the process identifier of the connected MATLAB instance. See ProcessID on page 57.
Version	String	To get the version of the connected MATLAB instance. See Version on page 58.
(Visible)	–	Discontinued as of dSPACE Release 2021-A.
WatchdogMethod	Int	To get or set the method for observing the MATLAB process. See WatchdogMethod on page 59.

Methods

The class Matlab provides the following methods:

Method	Purpose
Close	To end the connection to MATLAB and quit the MATLAB application. See Close on page 48.
Execute	To execute a MATLAB command. See Execute on page 51.
GetArray	To get a Python representation of a MATLAB array from a MATLAB workspace variable. See GetArray on page 51.
GetOutputs	To return the output of the last OBJ.Execute() command that ordinarily appears in the MATLAB Command Window. See GetOutputs on page 52.
IsAlive	To test if MATLAB is alive. See IsAlive on page 53.
(RestoreCommandWindow)	Discontinued as of dSPACE Release 2021-A.
(MaximizeCommandWindow)	Discontinued as of dSPACE Release 2021-A.
(MinimizeCommandWindow)	Discontinued as of dSPACE Release 2021-A.
Open	To open the connection to MATLAB. See Open on page 55.
PutArray	To assign an array or string to a MATLAB workspace variable. See PutArray on page 57.

Close

Class

Matlab

Syntax

```
OBJ.Close([DisconnectOnly = False])
```

Purpose

To end the connection to MATLAB and quit the MATLAB application.

Parameters

The method provides the following parameter.

Parameter	Type	Description
DisconnectOnly	Boolean	▪ True: The MATLAB instance remains open.

Parameter	Type	Description
		▪ False: The MATLAB instance closes if this client started the MATLAB instance and the MATLAB instance is not connected to another client. The default for this parameter is False.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or if MATLAB is not accessible.

Related methods

IsAlive on page 53,

Open on page 55

ConnectedMATLABInstallations

Class

Matlab

Syntax

```
[GetValue =] OBJ.ConnectedMATLABInstallations
```

Purpose

To get a list of the connected MATLAB installations containing the installation paths and whether they are configured as the preferred MATLAB instance.

Parameter

The attribute returns a value of the following type:

Type	Description
List of tuples	Each tuple consists of two values: ▪ String: Installation path of the connected MATLAB instance, e.g., `C:\Program Files\MATLAB\R2016a` ▪ Int: Shows whether the MATLAB installation is configured as the preferred MATLAB instance during installation or afterwards by using the dSPACE Installation Manager. 　▪ 0: MATLAB installation is not preferred. 　▪ 1: MATLAB installation is preferred.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

Version on page 58

ExecutablePath

Class

Matlab

Syntax

```
[GetValue =] OBJ.ExecutablePath
```

Purpose

To get the path to the executable of the connected MATLAB instance.

Parameters

The attribute returns a value of the following type:

Type	Description
String	Path to the executable of the connected MATLAB instance: e.g., `C:\Program Files\MATLAB\R2016a\bin\MATLAB.exe`.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

IsMUMatlabOpen on page 54

Version on page 58

Execute

Class	Matlab
Syntax	`OBJ.Execute(Command)`
Purpose	To execute a MATLAB command.
Description	Execute the MATLAB command specified by Command.

Parameter

The method uses the following parameter:

Parameter	Type	Description
Command	String Unicode	The MATLAB command to be executed.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	▪ The underlying COM call fails. ▪ MATLAB is not accessible. ▪ The MATLAB command fails for some reason. In that case the exception contains MATLAB's error message as string.

Related methods

GetOutputs on page 52

GetArray

Class	Matlab
Syntax	`[Result =] OBJ.GetArray(ArrayName)`

Purpose

To get a Python representation of a MATLAB array from a MATLAB workspace variable.

Description

Gets a Python representation of a MATLAB array from a MATLAB workspace variable. See Interfacing MATLAB (matlablib2) on page 23 for a list of supported MATLAB array classes and their Python representation. The name of the variable is given by `ArrayName`.

Parameter

The method uses the following parameter:

Parameter	Type	Description
ArrayName	String Unicode	Name of the MATLAB workspace variable

Return value

The method returns a value of the following type:

Type	Description
Various	A Python representation of a MATLAB array on success. See Interfacing MATLAB (matlablib2) on page 23 for a detailed list of supported MATLAB array classes and their conversion to an appropriate Python representation. None on error.

Exception

This method may raise the following exception:

Exception	Description
matlablibError	▪ The underlying COM call fails. ▪ MATLAB is not accessible.

Related methods

PutArray on page 57

GetOutputs

Class

Matlab

Syntax

```
[Result = ] OBJ.GetOutputs()
```

Purpose

To return the output of the last `Execute` command that ordinarily appears in the MATLAB Command Window.

Parameters

–

Return value

The method returns a value of the following type:

Type	Description
String	Output of the last `Execute` command

Exception

This method may raise the following exception:

Exception	Description
matlablibError	▪ The underlying COM call fails ▪ MATLAB is not accessible.

Related methods

Execute on page 51

IsAlive

Class

Matlab

Syntax

```
[Result =] OBJ.IsAlive()
```

Purpose

To test if MATLAB is alive.

Description

IsAlive() sends an "alive" message to MATLAB and tests if access to MATLAB still is possible.

Parameters

–

Return value

The method returns a value of the following type:

Type	Description
Boolean	A value different from 0 on success ("alive"), 0 on error ("dead").

Exception

–

Related methods

Close on page 48,

Open on page 55

IsMUMatlabOpen

Class

Matlab

Syntax

```
[GetValue =] OBJ.IsMUMatlabOpen
```

Purpose

To get the flag of whether the connected MATLAB instance is opened for multiple use.

Parameter

The attribute returns a value of the following type:

Type	Description
Tuple (Int, String, String)	▪ Int: If set to 1, the connected MATLAB instance is enabled for multiple use. ▪ String: Path to the executable of the connected MATLAB instance: e.g., `C:\Program Files\MATLAB\R2016a\bin\MATLAB.exe`. ▪ String: Version of the connected MATLAB instance: e.g., `R2016a`.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

ExecutablePath on page 50

Version on page 58

Open

Class

Matlab

Syntax

```
OBJ.Open([ConvertToDouble = 0], [StartNewMLInst = 0],
         [OpenFlags = 0], [StartCommands = ""],
         [MLStartupDir = ""], [MLInstallDir = ""])
```

Purpose

To open the connection to MATLAB.

Description

MATLAB starts if it is still not running.

Parameter

The method uses the following parameters:

Parameter	Type	Description
ConvertToDouble	Boolean	If this optional parameter is set to 1: ▪ All subsequent calls to PutArray convert Python integer types like int and long to the MATLAB type double. ▪ All subsequent calls to GetArray convert MATLAB integer types to Python float. The default for this parameter is 0 (false).
StartNewMLInst	Int	▪ 0: Connect to existing MATLAB instance ▪ 1: Start new MATLAB instance The default for this parameter is 0.
OpenFlags	Int	▪ matlablib2.Constants.ML_OPEN_FLAG_NOSPLASH: no splashscreen is displayed ▪ matlablib2.Constants.ML_OPEN_FLAG_NOJVM: disable java virtual machine ▪ matlablib2.Constants.ML_OPEN_FLAG_NODESKTOP: start MATLAB without desktop, MATLAB Command Window only The default for this parameter is 0 (no flag).
StartCommands	String	Command(s) to execute after starting MATLAB. The default for this parameter is an empty string.
MLStartupDir	String	Lets you specify the MATLAB startup directory. The default for this parameter is an empty string.
MLInstallDir	String	Lets you specify the MATLAB installation directory to start MATLAB from this directory. The default for this parameter is an empty string.

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related methods

Close on page 48,

IsAlive on page 53

ProcessArchitecture

Class

Matlab

Syntax

```
[GetValue =] OBJ.ProcessArchitecture
```

Purpose

To get the process architecture of the connected MATLAB instance.

Parameters

The attribute returns a value of the following type:

Type	Description
Int	▪ Constants.PROC_ARCHITECTURE_32BIT: 32-bit MATLAB instance ▪ Constants.PROC_ARCHITECTURE_64BIT: 64-bit MATLAB instance

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

ProcessID on page 57

ProcessID

Class

Matlab

Syntax

```
[GetValue =] OBJ.ProcessID
```

Purpose

To get the process identifier of the connected MATLAB instance.

Parameters

The attribute returns a value of the following type:

Type	Description
Int	Process ID of the connected MATLAB instance.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

ProcessArchitecture on page 56

PutArray

Class

Matlab

Syntax

```
OBJ.PutArray(ArrayName, Value)
```

Purpose

To assign an array or string to a MATLAB workspace variable.

Description

PutArray assigns an Array to a MATLAB workspace variable. The name of the variable is given by ArrayName. The second parameter specifies the Array (or Matrix) to be assigned to the variable. In Python, lists are used to represent a MATLAB array. See Interfacing MATLAB (matlablib2) on page 23 for the tables of type conversion rules.

Parameters

The method uses the following parameters:

Parameter	Type	Description
ArrayName	String Unicode	Name of the MATLAB workspace variable to that the value should be assigned to.
Value	Int, Float, List, String, Unicode, Dictionary, or CellArray	Value to be assigned to the MATLAB workspace variable with name "ArrayName".

Return value

–

Exception

This method may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related methods

GetArray on page 51

Version

Class

Matlab

Syntax

```
[GetValue =] OBJ.Version
```

Purpose

To get the version of the connected MATLAB instance.

Parameters

The attribute returns a value of the following type:

Type	Description
String	Version of the connected MATLAB instance: e.g., `R2016a`.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

ExecutablePath on page 50

IsMUMatlabOpen on page 54

WatchdogMethod

Class

Matlab

Syntax

```
OBJ.WatchdogMethod [= SetValue]
```

or

```
[GetValue =] OBJ.WatchdogMethod
```

Purpose

To set or get whether MATLAB restarts if the watchdog timer is exceeded.

Parameters

The attribute uses a value of the following type:

Type	Description
Integer	▪ Constants.WATCHDOG_PREVENT: MATLAB is prevented from being closed. ▪ Constants.WATCHDOG_RESTART: MATLAB will be restarted if necessary.

Exception

This attribute may raise the following exception:

Exception	Description
matlablibError	The underlying COM call fails or MATLAB is not accessible.

Related attributes

–

Examples (matlablib2)

Examples of Using matlablib2

Overview of the demo scripts

If you have installed dSPACE Python Extensions, you will find the following example scripts in:
`<dSPACEPythonExtensions_InstallationFolder>\Demos\Python Test Automation\Interfacing Matlab (matlablib2)`:

- `ExecutingCommands\d_ExecuteMatlabCommands.py`

  This script creates arrays in MATLAB whose contents are plotted using the MATLAB plot command. Also, the observation of MATLAB results from Python is demonstrated.
- `ExchangeOfSimpleDatatypes\d_SimpleDataTypes.py`

  This script generates arrays of different types and sizes, puts them to MATLAB's workspace and reads them out again. It concentrates on the more simple datatypes in MATLAB. Use the `whos` command in MATLAB to observe the results.
- `ExchangeOfComplexDatatypes\d_ComplexDataTypes.py`

  This script generates arrays of different types and sizes, puts them to MATLAB's workspace and reads them out again. It concentrates on the more complex datatypes CellArray and StructArray.
- `ReadingAndWritingMatfiles\d_ReadAndWriteMatFiles.py`

  This script demonstrates how to read data from and write data to MAT files. It creates a MAT file which is then read again to copy the contents into another MAT file. Finally, two identical MAT files exist.

To start the demo scripts, use an external Python interpreter, for example, PythonWin. You find it in Python 3.9 - PythonWin in the Windows Start menu.

Before you start a script, read its description for further information.

Tip

You can run the demo script also in AutomationDesk by using an ExecFile block.

Related topics

References

ExecFile (AutomationDesk Basic Practices 📖)
Interfacing MATLAB (matlablib2)... 23

Multiple Use of Identifiers Within the Test Automation Python Modules

Multiply used identifiers

In the Test Automation Python Modules, some method and class identifiers are multiply used. This does not mean that the identifiers are ambiguous. The identifiers are unique, because each object must be fully qualified.

The table below gives an overview of the identifiers that are multiply used in the Test Automation Python Modules.

The following topics are intended to facilitate access to the related documentation in dSPACE Help. For each multiply used identifier there is an overview of the different contexts and quick access to the specific help topics.

Identifier	Type	Concerned dSPACE Python Module
Close on page 62	Method	matlablib2, rs232lib2
Open on page 62	Method	matlablib2, rs232lib2

Where to go from here

Information in this section

Close

Context of the identifier

The identifier is used in the following contexts:

Module	Context	Detailed Information
	Interfacing MATLAB (matlablib2)	
matlablib2	Matfile class	See Close on page 37.
matlablib2	Matlab class	See Close on page 48.
	Acquiring Data from External Devices rs232lib2)	
rs232lib2	rs232lib2 function	See Close on page 12.

GetArray

Context of the identifier

The identifier is used in the following contexts:

Module	Context	Detailed Information
	Interfacing MATLAB (matlablib2)	
matlablib2	Matfile class	See GetArray on page 38.
matlablib2	Matlab class	See GetArray on page 51.

Open

Context of the identifier

The identifier is used in the following contexts:

Module	Context	Detailed Information
	Interfacing MATLAB (matlablib2)	
matlablib2	Matfile class	See Open on page 41.
matlablib2	Matlab class	See Open on page 55.
	Acquiring Data from External Devices (rs232lib2)	
rs232lib2	rs232lib2 function	See Open on page 14.

PutArray

Context of the identifier

The identifier is used in the following contexts:

Module	Context	Detailed Information
	Interfacing MATLAB (matlablib2)	
matlablib2	Matfile class	See PutArray on page 42.
matlablib2	Matlab class	See PutArray on page 57.