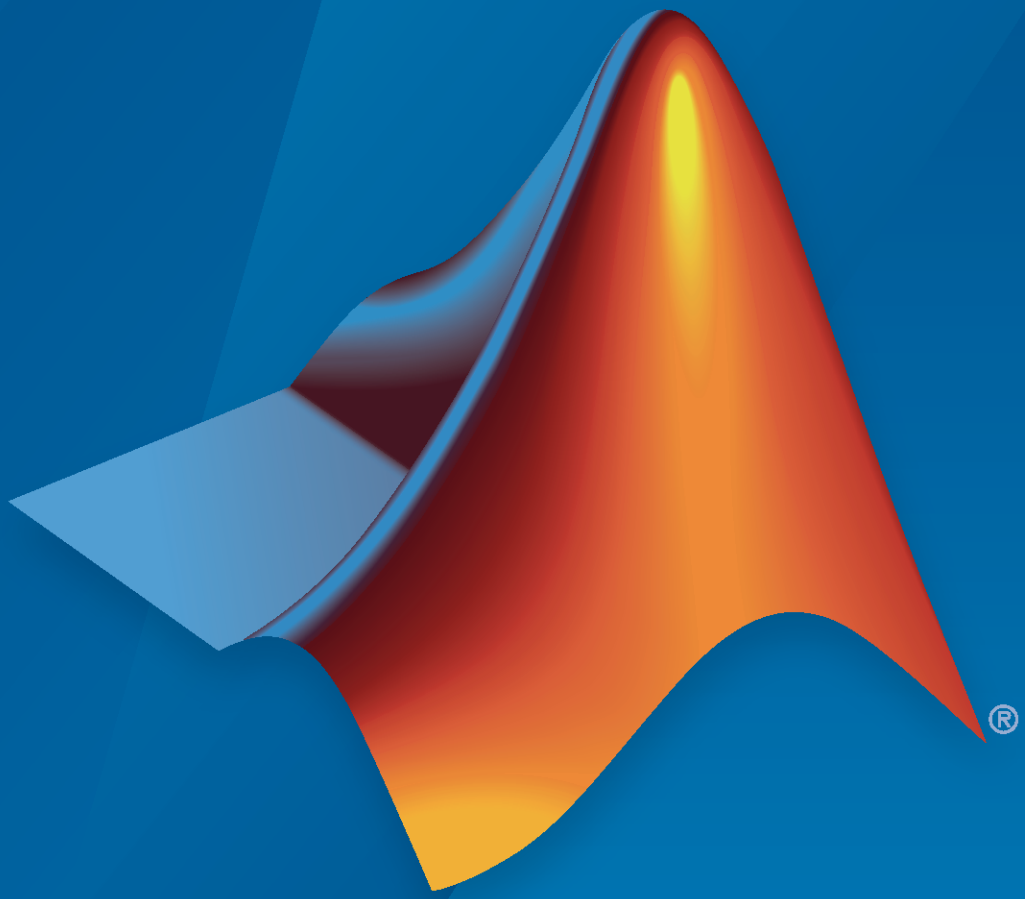
MATLAB® Production Server™

Python® Client Programming



MATLAB®

R2022b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

MATLAB® Production Server™ Python® Client Programming

© COPYRIGHT 2012–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

October 2014	Online only	New for Version 2.0 (Release R2014b)
March 2015	Online only	Revised for Version 2.1 (Release R2015a)
September 2015	Online only	Revised for Version 2.2 (Release R2015b)
March 2016	Online only	Revised for Version 2.3 (Release 2016a)
September 2016	Online only	Revised for Version 2.4 (Release 2016b)
March 2017	Online only	Revised for Version 3.0 (Release 2017a)
September 2017	Online only	Revised for Version 3.0.1 (Release R2017b)
March 2018	Online only	Revised for Version 3.1 (Release R2018a)
September 2018	Online only	Revised for Version 4.0 (Release R2018b)
March 2019	Online only	Revised for Version 4.1 (Release R2019a)
September 2019	Online only	Revised for Version 4.2 (Release R2019b)
March 2020	Online only	Revised for Version 4.3 (Release R2020a)
September 2020	Online only	Revised for Version 4.4 (Release R2020b)
March 2021	Online only	Revised for Version 4.5 (Release R2021a)
September 2021	Online only	Revised for Version 4.6 (Release R2021b)
March 2022	Online only	Revised for Version 5.0 (Release R2022a)
September 2022	Online only	Revised for Version 5.1 (Release R2022b)

Contents

1 Client Programming

2 Python Client Development

3 Data Handling

4 APIs

Client Programming

Create a MATLAB Production Server Python Client

You can call a MATLAB function deployed to MATLAB Production Server from a Python client application. To create a Python client:

1. Install the MATLAB Production Server client runtime files.

   For details, see “Install the MATLAB Production Server Python Client” on page 2-2.
2. In consultation with the MATLAB programmer, collect the MATLAB function signatures that comprise the services in the application.
3. Write Python code to instantiate a connection to a MATLAB Production Server instance.

   For different ways to create a connection, see “Create Client Connection” on page 2-3.
4. Create the required MATLAB data for function inputs and outputs.

   For using arrays as function arguments, see “matlab Python Module” on page 3-4. For other data types, see “Pass Data Between MATLAB Production Server and Python” on page 3-2.
5. Evaluate the MATLAB functions.

   For more information about ways to call deployed MATLAB functions, see “Invoke Packaged MATLAB Functions” on page 2-5.
6. Close the client connection.

See Also

`matlab.production_server.client.MWHttpClient`

Related Examples

- “Create a Python Client”
- “Create Deployable Archive for MATLAB Production Server”

Create a Python Client

This example shows how to write a MATLAB Production Server client using the Python client API. The client application calls the `addmatrix` MATLAB function deployed to a server instance. For information on writing and compiling the function for deployment, see “Create Deployable Archive for MATLAB Production Server”. For deploying the function to the server, see “Deploy Archive to MATLAB Production Server”.

Before you write the client application, you must have the MATLAB Production Server Python client libraries installed on your system. For details, see “Install the MATLAB Production Server Python Client” on page 2-2.

1. Start the Python command line interpreter.
2. Enter the following import statements at the Python command prompt.

   ```
   import matlab
   from production_server import client
   ```
3. Open the connection to the MATLAB Production Server instance and initialize the client runtime.

   ```
   client_obj = client.MWHttpClient("http://localhost:9910")
   ```
4. Create the MATLAB data to input to the function.

   ```
   a1 = matlab.double([[1,2,3],[3,2,1]])
   a2 = matlab.double([[4,5,6],[6,5,4]])
   ```
5. Call the deployed MATLAB function. To call the function, you must know the name of the deployed archive and the name of the function.

   The syntax for invoking a function is `client.`*`archiveName`*`.`*`functionName`*`(`*`arg1`*`, `*`arg2`*`, .., [nargout=`*`numOutArgs`*`])`.

   ```
   client_obj.addmatrix.addmatrix(a1,a2)
   ```

   The output is:

   ```
   matlab.double([[5.0,7.0,9.0],[9.0,7.0,5.0]])
   ```
6. Close the client connection.

   ```
   client_obj.close()
   ```

See Also

`matlab.production_server.client.MWHttpClient`

Related Examples

- “Create Client Connection” on page 2-3
- “Invoke Packaged MATLAB Functions” on page 2-5

Python Client Development

Install the MATLAB Production Server Python Client

In this section...

The MATLAB Production Server client APIs are available for download at MATLAB Production Server Client Libraries. In an on-premises MATLAB Production Server installation, the client APIs are located in *MPS_INSTALL*`/client`, where `$MPS_INSTALL` is the MATLAB Production Server installation location.

Supported Python Interpreters

For information about versions of Python that the MATLAB Production Server Python client supports, see Product Requirements & Platform Availability for MATLAB Production Server.

Installation Procedure

The MATLAB Production Server Python client provides a standard Python setup script. This script installs the required modules into your Python environment.

1. Navigate to the Python client API folder.

   Example 2.1. UNIX

   ```
   cd MPS_INSTALL/client/python
   ```

   Example 2.2. Windows

   ```
   cd MPS_INSTALL\client\python
   ```

2. Run the setup script. You require write and execute permissions in the directory where you run the script.

   ```
   python setup.py install
   ```

See Also

More About

- “Create a Python Client” on page 1-3
- “Create a MATLAB Production Server Python Client” on page 1-2

Create Client Connection

In this section...

The connection between a Python client and a MATLAB Production Server instance is encapsulated in a `matlab.production_server.client.MWHttpClient` object. You use the `MWHttpClient` constructor to instantiate the connection between the client and the server.

The `MWHttpClient()` constructor has the following signature:

```
client.MWHttpClient(url[,timeout_ms=timeout,ssl_context=ssl_context])
```

The constructor has the following arguments:

- *`url`* — URL of the server instance to which the client connects. If the URL is to an on-premises server instance, the URL must contain the port number of the server instance.

  Note The URL contains only the host name and port information of the server instance.

- *`timeout_ms`* — Amount of time, in milliseconds, that the client waits for a response before timing out.

  The default time-out interval is two minutes.

- *`ssl_context`* — `ssl.SSLContext` object that contains information about the SSL protocol to use for HTTPS communication with the server. If the URL of the server instance contains HTTPS, this argument is required.

  The default is to not use SSL.

Note The `MWHttpClient` object is not thread-safe. If you are developing a multithreaded application, create a new `MWHttpClient` object for each thread.

Create Default Connection

To create a default connection, provide a value for the server instance URL. The `timeout_ms` argument has a default value, so you do not need to specify a time. The default is to use HTTP for client-server communication. This code sample shows how to connect to server instance on a host named `mps_host` using the default time-out of two minutes.

```
import matlab
from production_server import client

my_client = client.MWHttpClient("http://mps_host:9910")
```

Configure Connection Timeout

You specify the connection time out by providing a value for the `timeout_ms` argument. This code sample specifies a time-out of one minute.

```
import matlab
from production_server import client

my_client = client.MWHttpClient("http://mps_host:9910",timeout_ms=60000)
```

Use HTTPS for Client-Server Communication

The MATLAB Production Server Python client API uses the Python `ssl` library for supporting HTTPS communication with the server. You specify SSL connection properties by providing an object of the Python `ssl.SSLContext` class as value for the `ssl_context` argument. You can pass a parameter to the `ssl.SSLContext` object to set the SSL protocol to use. For more information about the SSL protocols that the server supports, see `ssl-protocols`.

HTTPS communication using the Python client API is supported only on Windows® and Linux® platforms. Mac OS is not supported.

This code sample sets the SSL protocol to `PROTOCOL_TLS_CLIENT`. Setting the protocol to `PROTOCOL_TLS_CLIENT` requires you to provide details about the SSL certificate of the server.

```
import ssl
import matlab
from production_server import client

context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations(cafile='<path_to_server_SSL_certificate_location>\cert_file.pem')
my_client = client.MWHttpClient("https://mps_host:9920",ssl_context=context)
```

See Also

`matlab.production_server.client.MWHttpClient`

Related Examples

- “Create a Python Client”
- “Invoke Packaged MATLAB Functions” on page 2-5
- “Handle Function Processing Errors” on page 2-7

External Websites

- class ssl.SSLContext
- Python `ssl` library

Invoke Packaged MATLAB Functions

In this section...

The connection between a Python client and a MATLAB Production Server instance is encapsulated in a `matlab.production_server.client.MWHttpClient` object. You invoke deployed MATLAB functions using the client connection object.

```
result1,...resultN = my_client.archive_name.function_name(in_args,
                                                          nargout=nargs)
```

- *`my_client`* — Name of client connection object
- *`archive_name`* — Name of the deployable archive hosting the function
- *`function_name`* — Name of the function to invoke
- *`in_args`* — Comma-separated list of input arguments
- *`nargs`* — Number of results expected from the server. The default value is 1.

Note If the function to invoke returns an output, each variable on the left side of the function call is populated with a single return value. If you provide less than `nargs` variables on the left side of the function call, the last listed variable contains a list of the remaining results. For example,

```
result1, result2 = myMagic.triple(5,nargout=3)
```

leaves `result1` containing a single value and `result2` containing a list with two values.

Invoke MATLAB Functions that Return Multiple Outputs

Receive Multiple Results as Individual Variables

To invoke the MATLAB function `c1,c2 = copy(o1,o2)` from the deployable archive `copier`, use this code:

```
>>> import matlab
>>> from production_server import client
>>> my_client = client.MWHttpClient("http://localhost:9910")
>>> c1,c2 = my_client.copier.copy("blue",10,nargout=2)
>>> print(c1)
"blue"
>>> print(c2)
10
```

The variables `c1` and `c2` are populated with a single return value.

Receive Multiple Results as Single Object

To invoke the MATLAB function `copies = copy(o1,o2)` from the deployable archive `copier`, use this code:

```
>>> import matlab
>>> from production_server import client
>>> my_client = client.MWHttpClient("http://localhost:9910")
>>> copies = my_client.copier.copy("blue",10,nargout=2)
>>> print(copies)
["blue",10]
```

The variable `copies` is populated with a list containing all of the returned values.

Invoke MATLAB Functions that Return Zero Outputs

To invoke the MATLAB function `mutate(m1, m2, m3)` from the deployable archive `mutations`, you use this code:

```
import matlab
from production_server import client

my_client = client.MWHttpClient("http://localhost:9910")

m1 = matlab.double(...)
m2 = matlab.double(...)
m3 = matlab.double(...)

my_client.mutations.mutate(m1,m2,m3)
```

Invoke MATLAB Functions that Return Single Output

To invoke the MATLAB function `result = mutate(m1, m2, m3)` from the deployable archive `mutations`, you use this code:

```
import matlab
from production_server import client

my_client = client.MWHttpClient("http://localhost:9910")

m1 = matlab.double(...)
m2 = matlab.double(...)
m3 = matlab.double(...)

result = my_client.mutations.mutate(m1,m2,m3)
```

See Also

`matlab.production_server.client.MWHttpClient`

Related Examples

- “Create a Python Client”
- “matlab Python Module” on page 3-4

Handle Function Processing Errors

In this section...

The common types of exceptions that can occur when evaluating MATLAB functions include:

- HTTP errors — Handled using the Python `httplib.HTTPException` exception. Common reasons for HTTP errors include:
  - Using an incorrect archive name
  - Using an incorrect function name
  - Timing out before the function finishes evaluating
- MATLAB Runtime errors — Handled using the `matlab.mpsexception.MATLABException` exception. Occurs when the MATLAB Runtime generates an error while evaluating a function.

Your client code should handle these errors gracefully.

HTTP Errors

If your client code experiences any issues when sending data to or receiving data from a server instance, an `httplib.HTTPException` exception is raised. A common cause for an HTTP error is a name mismatch between deployed artifacts on the server and the functions called in the client.

For example, deploying the function `mutate()` in the archive `mutations` the following results in an error because the server instance would not be able to resolve the name of the archive.

```
import httplib
import matlab
from production_server import client

def main()
  my_client = client.MWHttpClient("http://localhost:9190")

  try:
    result = my_client.mutation.mutate("blue",10,12)
    ...
  except httplib.HTTPException as e:
    print e
```

If you deploy the function `mutate()` in the archive `mutations`, the following results in an error because the server instance would not be able to resolve the name of the function.

```
import httplib
import matlab
from production_server import client

def main()
  my_client = client.MWHttpClient("http://localhost:9190")

  try:
    result = my_client.mutations.mutator("blue",10,12)
```

```
    ...
  except httplib.HTTPException as e:
    print e
```

MATLAB Runtime Errors

If an error occurs while the MATLAB Runtime is evaluating a function, a `matlab.mpsexception.MATLABException` exception is raised. The exception contains the following:

- `ml_error_message` — Error message returned by the MATLAB Runtime
- `ml_error_identifier` — MATLAB error ID
- `ml_error_stack` — MATLAB Runtime stack

This function catches any MATLAB Runtime errors and prints them to the console.

```
from matlab.production_server import client
from matlab.production_server import mpsexceptions
import sys

def main(size):

  my_client = client.MWHttpClient('http://localhost:9190')
  try:
    data = my_client.magic.mymagic(size)
    print data
  except mpsexceptions.MATLABException as e:
    print 'MATLAB Error: ',e

  my_client.close()
```

See Also

`matlab.production_server.client.MWHttpClient`

Related Examples

- “Create a Python Client”
- “Create Client Connection” on page 2-3

Data Handling

Pass Data Between MATLAB Production Server and Python

In this section...

Pass Data from MATLAB Production Server to Python

When MATLAB functions return output arguments, MATLAB Production Server converts the data into equivalent Python data types.

MATLAB Output Argument Type (scalar unless otherwise noted)	Resulting Python Data Type
Numeric array	`matlab` numeric array object (see “matlab Python Module” on page 3-4)
`double`, `single`	`float`
Complex (any numeric type)	`complex`
`int8`, `uint8`, `int16`, `uint16`, `int32`	`int`
`uint32`, `int64`, `uint64`	`int`
`NaN`	`float ('nan')`
`Inf`	`float ('inf')`
`logical`	`bool`
`char` array (1-by-`N`, `N`-by-1) `char` array (`M`-by-`N`)	`str` Not supported
structure	`dict`
Row or column cell array	`list`
`M`-by-`N` cell array	Not supported
MATLAB handle object (such as the `containers.Map` type)	`matlab.object` MATLAB returns a reference to a `matlab.object`, not the object itself. You cannot pass a `matlab.object` between MATLAB sessions.
MATLAB value object (such as the `categorical` type)	Opaque object. You can pass a value object to a MATLAB function, but you cannot create or modify it.
Other object (for example, Java® object)	Not supported
Function handle	Not supported
Sparse array	Not supported
String array	Not supported
Structure array	Not supported

Pass Data from Python to MATLAB Production Server

When you pass data as input arguments to MATLAB functions from Python, MATLAB Production Server converts the data into equivalent MATLAB data types.

Python Input Argument Type	Resulting MATLAB Data Type (scalar unless otherwise noted)
`matlab` numeric array object (see "matlab Python Module" on page 3-4)	Numeric array
`float`	`double`
`complex`	Complex `double`
`int`	`int32`(Windows) `int64`(Linux and Mac)
`float('nan')`	`NaN`
`float('inf')`	`Inf`
`bool`	`logical`
`str`	`char`
`bytearray`	`uint8` array
`bytes`	`uint8` array
`dict`	Structure if all keys are strings. Not supported otherwise
`list`	Cell array
`set`	Cell array
`tuple`	Cell array
`memoryview`	Not supported
`range`	Cell array
`None`	Not supported
`module.type`	Not supported

See Also

Related Examples

- "Use MATLAB Arrays in Python" on page 3-10
- "matlab Python Module" on page 3-4
- "Invoke Packaged MATLAB Functions" on page 2-5

matlab Python Module

In this section...

The `matlab` Python module provides array classes to represent arrays of MATLAB numeric types as Python variables so that MATLAB arrays can be passed between Python and MATLAB.

MATLAB Classes in the matlab Python Module

- You can use MATLAB numeric arrays in Python code by importing the `matlab` Python package and calling the necessary constructors. For example:

  ```
  import matlab
  a = matlab.double([[1, 2, 3],[4, 5, 6]])
  ```

  The name of the constructor indicates the MATLAB numeric type. You can pass MATLAB arrays as input arguments to MATLAB functions called from Python. When a MATLAB function returns a numeric array as an output argument, the array is returned to Python.
- You can initialize an array with an optional `initializer` input argument that contains numbers. The `initializer` argument must be a Python sequence type such as a `list`, `tuple`, or `range`. You can specify `initializer` to contain multiple sequences of numbers.
- You can initialize an array with an optional `vector` input argument that contains input of size 1-by-N. If you use `vector`, you cannot use `initializer`.
- You can create a multidimensional array using one of the following options:
  - Specify a nested sequence without specifying the size.
  - Specify a nested sequence and also specify a `size` input argument that matches the dimensions of the nested sequence.
  - Specify a one-dimensional sequence together with a multidimensional size. In this case, the sequence will be assumed to represent the elements in column-major order.
- You can create a MATLAB array of complex numbers by setting the optional `is_complex` keyword argument to `True`.
- You can use custom types for initializing MATLAB arrays in Python. The custom type should implement the Python buffer protocol. One example is `ndarray` in NumPy.

Class from `matlab` Python Package	Constructor Call in Python

`matlab.double`	`matlab.double(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.single`	`matlab.single(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.int8`	`matlab.int8(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.int16`	`matlab.int16(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.int32`	`matlab.int32(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.int64`	`matlab.int64(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.uint8`	`matlab.uint8(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.uint16`	`matlab.uint16(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.uint32`	`matlab.uint32(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.uint64`	`matlab.uint64(initializer=None|vector=None,`
	`size=None,`
	`is_complex=False)`

`matlab.logical`	`matlab.logical(initializer=None|vector=None,` `size=None)`[a]

a Logicals cannot be made into an array of complex numbers.

Properties and Methods of MATLAB Classes in the matlab Python Package

All MATLAB arrays created with package constructors have the following properties and methods:

Properties

Property Name	Description	Examples

`size`	A tuple of integers representing the dimensions of an array	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> a.size`
		`(2, 3)`

`itemsize`	An integer representing the size in bytes of an element of the array	`>>> a = matlab.int16()`
		`>>> a.itemsize`
		`2`
		`>>> b = matlab.int32()`
		`>>> b.itemsize`
		`4`

Methods

Method Name	Purpose	Examples

`clone()`	Return a new distinct object with contents identical to the contents of the original object	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> b = a.clone()`
		`>>> print(b)`
		`[[1,2,3],[4,5,6]]`
		`>>> b[0][0] = 100`
		`>>> b matlab.int16([[100,2,3],[4,5,6]])`
		`>>> print(a )`
		`[[1,2,3],[4,5,6]]`

`real()`	Return the real parts of elements that are complex numbers, in column-major order, as a 1-by-N array	`>>> a = matlab.int16([[1 + 10j, 2 + 20j`
		`>>> print(a.real())`
		`[1,4,2,5,3,6]`

`imag()`	Return the imaginary parts of elements that are complex numbers, in column-major order, as a 1-by-N array	`>>> a = matlab.int16([[1 + 10j, 2 + 20j`
		`>>> print(a.imag())`
		`[10,0,20,0,30,0]`

`noncomplex()`	Return elements that are not complex numbers, in column-major order, as a 1-by-N array	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> print(a.noncomplex())`
		`[1,4,2,5,3,6]`

• `reshape(`*`dim1`*`,`*`dim2`*`,...,`*`dimN`*`)`
• `reshape((`*`dim1`*`,`*`dim2`*`,...,`*`dimN`*`))`
• `reshape([`*`dim1`*`,`*`dim2`*`,...,`*`dimN`*`])`
	Reshape the array according to the dimensions and return the result	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> print(a)`
		`[[1,2,3],[4,5,6]]`
		`>>> a.reshape(3, 2)`
		`>>> print(a)`
		`[[1,5],[4,3],[2,6]]`

`toarray()`	Return a standard Python `array.array` object constructed from the contents. Applicable for one-dimensional sequences only.	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> a[0].toarray()`
		`array('h', [1, 2, 3])`
		`>>> b = matlab.int16([[1 + 10j, 2 + 20j`
		`>>> b.real().toarray()`
		`array('h', [1, 4, 2, 5, 3, 6])`

`tomemoryview()`	Return a standard Python `memoryview` object constructed from the contents	`>>> a = matlab.int16([[1, 2, 3],[4, 5,`
		`>>> b = a.tomemoryview()`
		`>>> b.tolist()`
		`[[1, 2, 3], [4, 5, 6]]`
		`>>> b.shape`
		`(2, 3)`

Create a MATLAB Array with N Elements

When you create an array with `N` elements, the size is 1-by-`N` because it is a MATLAB array.

```
import matlab
A = matlab.int8([1,2,3,4,5])
print(A.size)

(1, 5)
```

The initializer is a Python list containing five numbers. The MATLAB array size is 1-by-5, indicated by the tuple `(1,5)`.

Multidimensional MATLAB Arrays in Python

In Python, you can create multidimensional MATLAB arrays of any numeric type. Use a nested Python list of floats to create a 2-by-5 MATLAB array of doubles.

```
import matlab
A = matlab.double([[1,2,3,4,5], [6,7,8,9,10]])
print(A)

[[1.0,2.0,3.0,4.0,5.0],[6.0,7.0,8.0,9.0,10.0]]
```

The `size` attribute of `A` shows it is a 2-by-5 array.

```
print(A.size)

(2, 5)
```

Index Into MATLAB Arrays in Python

You can index into MATLAB arrays just as you can index into Python lists and tuples.

```
import matlab
A = matlab.int8([1,2,3,4,5])
print(A[0])

[1,2,3,4,5]
```

The size of the MATLAB array is `(1,5)`; therefore, `A[0]` is `[1,2,3,4,5]`. Index into the array to get 3.

```
print(A[0][2])

3
```

Python indexing is zero-based. When you access elements of MATLAB arrays in a Python session, use zero-based indexing.

This example shows how to index into a multidimensional MATLAB array.

```
A = matlab.double([[1,2,3,4,5], [6,7,8,9,10]])
print(A[1][2])

8.0
```

Slice MATLAB Arrays in Python

You can slice MATLAB arrays just as you can slice Python lists and tuples.

```
import matlab
A = matlab.int8([[1,2,3,4,5]])
print(A[0][1:4])

[2,3,4]
```

You can assign data to a slice. This example shows an assignment from a Python list to the array.

```
A = matlab.double([[1,2,3,4],[5,6,7,8]])
A[0] = [10,20,30,40]
print(A)

[[10.0,20.0,30.0,40.0],[5.0,6.0,7.0,8.0]]
```

You can assign data from another MATLAB array, or from any Python iterable that contains numbers.

You can specify slices for assignment, as shown in this example.

```
A = matlab.int8([1,2,3,4,5,6,7,8])
A[0][2:4] = [30,40]
A[0][6:8] = [70,80]
print(A)

[[1,2,30,40,5,6,70,80]]
```

Reshaping MATLAB Arrays in Python

You can reshape a MATLAB array in Python with the `reshape` method. The input argument, `size`, must be a sequence that does not change the number of elements in the array. Use `reshape` to change a 1-by-9 MATLAB array to 3-by-3. Elements are taken from the original array in column-major order.

```
import matlab
A = matlab.int8([1,2,3,4,5,6,7,8,9])
A.reshape((3,3))
print(A)

[[1,4,7],[2,5,8],[3,6,9]]
```

Use Custom Types to Initialize MATLAB Arrays

You can use custom types such as the `ndarray` in NumPy for initializing MATLAB arrays in Python. The custom type should implement the Python buffer protocol.

```
import matlab
import numpy

nf = numpy.array([[1.1, 2,2, 3.3], [4.4, 5.5, 6.6]])
md = matlab.double(nf)
ni32 = numpy.array([[1, 2, 3], [4, 5, 6]], dtype='int32')
mi32 = matlab.int32(ni32)
```

See Also

Related Examples

- “Use MATLAB Arrays in Python” on page 3-10
- “Pass Data to MATLAB from Python” (MATLAB)

Use MATLAB Arrays in Python

This example shows how to use MATLAB arrays in Python.

The `matlab` package provides new Python data types to create arrays that can be passed to MATLAB functions. The `matlab` package can create arrays of any MATLAB numeric or logical type from Python sequence types. Multidimensional MATLAB arrays are supported.

Create a MATLAB array in Python, and call a MATLAB function on it.

```
import matlab
from production_server import client
client_obj = client.MWHttpClient("http://localhost:9910")
x = matlab.double([1,4,9,16,25])
print(client_obj.myArchive.sqrt(x))

[[1.0,2.0,3.0,4.0,5.0]]
```

You can use `matlab.double` to create an array of doubles given a Python list that contains numbers. You can call a MATLAB function such as `sqrt` on `x`, and the return value is another `matlab.double` array.

Create a multidimensional array. The `magic` function returns a 2-D array to Python scope.

```
a = client_obj.myArchive.magic(6)
print(a)

[[35.0,1.0,6.0,26.0,19.0,24.0],[3.0,32.0,7.0,21.0,23.0,25.0],
 [31.0,9.0,2.0,22.0,27.0,20.0],[8.0,28.0,33.0,17.0,10.0,15.0],
 [30.0,5.0,34.0,12.0,14.0,16.0],[4.0,36.0,29.0,13.0,18.0,11.0]]
```

Call the `tril` function to get the lower triangular portion of `a`.

```
b = client_obj.myArchive.tril(a)
print(b)

[[35.0,0.0,0.0,0.0,0.0,0.0],[3.0,32.0,0.0,0.0,0.0,0.0],
 [31.0,9.0,2.0,0.0,0.0,0.0],[8.0,28.0,33.0,17.0,0.0,0.0],
 [30.0,5.0,34.0,12.0,14.0,0.0],[4.0,36.0,29.0,13.0,18.0,11.0]]
```

See Also

More About

- “matlab Python Module” on page 3-4

APIs

matlab.production_server.client.MWHttpClient

Package: `matlab.production_server`

Python object encapsulating a connection to a MATLAB Production Server instance

Description

The `matlab.production_server.client.MWHttpClient` class creates a connection object that encapsulates the connection between the client and a MATLAB Production Server instance. Once the connection is created, you can dynamically call all MATLAB functions hosted on the server instance.

Construction

```
my_client = MWHttpClient(url,[timeout_ms=timeout_ms],[ssl_context=
ssl_context])
```

Input Arguments

`url` — URL of the server instance to connect to

string

URL of the server instance to which the client connects, specified as a string. This server instance hosts the MATLAB functions which the client can evaluate.

`timeout_ms` — number of milliseconds the client waits for a response from the server instance

120000 (default)

Number of milliseconds the client waits for a response from the server instance, specified as an integer.

`ssl_context` — `SSLContext` object that specifies the SSL protocol to use for client-server communication

`None` (default) | `ssl.SSLContext` object

SSL protocol to use for client-server communication, specified as an `ssl.SSLContext` object. The Python client library uses the Python `ssl` library library for supporting HTTPS requests to server instances. For information about the SSL protocols that the server supports, see `ssl-protocols`.

This argument is required if the URL to connect to the server instance uses HTTPS.

HTTPS communication using the Python client API is supported only on Windows and Linux platforms. Mac OS is not supported.

Methods

Exceptions

`HTTPException`	Raised if there is a problem communicating with the server instance.
`MATLABException`	Raised if a function call fails to execute.
`TypeError`	Raised if the specified timeout value is not a positive `int` or `long`.
`ValueError`	Raised if the specified timeout value is less than zero.

Version History

Support for Python 2.7 will be discontinued in R2022b
Not recommended starting in R2022a

The MATLAB Production Server Python client library will not support Python 2.7 in future releases. If you want to use Python 2.7 to develop client applications, you can continue using the R2022a version of the Python client library in future releases.

For more information on client system requirements, see Product Requirements and Platform Availability for MATLAB Production Server.

Python 2.7 no longer supported in R2022b
Errors starting in R2022b

The MATLAB Production Server Python client library no longer supports Python 2.7. If you want to use Python 2.7 to develop client applications, you can continue using the R2022a version of the Python client library in future releases.

For more information on client system requirements, see Product Requirements and Platform Availability for MATLAB Production Server.

See Also

Topics
“Create Client Connection” on page 2-3
“Invoke Packaged MATLAB Functions” on page 2-5

External Websites
ssl.SSLContext
Python `ssl` library