MATLAB® Compiler SDK™

MATLAB® Production Server™ Python® Client Guide

MATLAB®

R2018a

MathWorks®

How to Contact MathWorks

MATLAB® Compiler SDK™ MATLAB® Production Server™ Python® Client Guide

Revision History

March 2015	Online only	New for Version 6.0 (Release R2015a)
September 2015	Online only	Revised for Version 6.1 (Release 2015b)
March 2016	Online only	Revised for Version 6.2 (Release 2016a)
September 2016	Online only	Revised for Version 6.3 (Release R2016b)
March 2017	Online only	Revised for Version 6.3.1 (Release R2017a)
September 2017	Online only	Revised for Version 6.4 (Release R2017b)
March 2018	Online only	Revised for Version 6.5 (Release R2018a)

Contents

Client Programming

MATLAB Production Server Examples

Additional Client examples for MATLAB Production Server are available in the `client` folder of your MATLAB Production Server.

Create a MATLAB Production Server Python Client

To create a MATLAB Production Server client:

1. Install the client run-time files.

   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 the Python code to instantiate a connection to a MATLAB Production Server instance.

   See “Create Client Connection” on page 2-3
4. Create the required MATLAB data for function inputs and outputs.

   See “MATLAB Arrays as Python Variables” on page 3-5.
5. Evaluate the MATLAB functions.

   See “Invoke MATLAB Functions that Return a Single Output” on page 2-6 or “Invoke MATLAB Functions that Return Multiple Outputs” on page 2-7
6. Close the client connection.

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` function you compiled in “Create a Deployable Archive for MATLAB Production Server” (MATLAB Production Server) and deployed in “Share a Deployable Archive on the Server Instance” (MATLAB Production Server).

Create a Python MATLAB Production Server client application:

1. Copy the contents of the `*matlabroot*\toolbox\compiler_sdk\mps_clients\python` folder to your development environment.
2. Open a command line,
3. Change directories into the folder where you copied the MATLAB Production Server Python client.
4. Run the following command.

   ```
   python setup.py install
   ```
5. Start the Python command line interpreter.
6. Enter the following import statements at the Python command prompt.

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

   ```
   client_obj = client.MWHttpClient("http://localhost:9910")
   ```
8. 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]])
   ```
9. Call the deployed MATLAB function.

   You must know the following:

   - Name of the deployed archive
   - Name of the function

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

   matlab.double([[5.0,7.0,9.0],[9.0,7.0,5.0]])
   ```

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

10 Close the client connection.

```
client_obj.close()
```

Unsupported MATLAB Data Types for Client and Server Marshaling

These data types are not supported for marshaling between MATLAB Production Server instances and clients:

- MATLAB function handles
- Complex (imaginary) data
- Sparse arrays

Python Client Development

Install the MATLAB Production Server Python Client

In this section...

Supported Python Interpreters

The MATLAB Production Server Python client supports Python v. 2.7 (https://www.python.org/download/releases/2.7.7/).

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. Change into the Python client folder.

   Example 2.1. UNIX

   ```
   cd matlabroot/toolbox/compiler_sdk/mps_clients/python
   ```

   Example 2.2. Windows

   ```
   cd matlabroot\toolbox\compiler_sdk\mps_clients\python
   ```
2. Run the setup script.

   ```
   python setup.py install
   ```

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 constructor to instantiate the connection between the client and the server.

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

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

The constructor has the following arguments:

- *`url`* — URL of the server instance to which the client connects. 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.

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

Create a 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. 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 the Connection Time Out

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)
```

Invoke MATLAB Functions that Return Zero Outputs

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

`void = *my_client*.*archive_name*.*function_name*(*in_args*, nargout=0)`

- *`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

For example, 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)
```

See Also

Related Examples

- “Invoke MATLAB Functions that Return a Single Output” on page 2-6
- “Invoke MATLAB Functions that Return Multiple Outputs” on page 2-7

Invoke MATLAB Functions that Return a Single Output

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

result = *my_client*.*archive_name*.*function_name*(*in_args*)

- *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

For example, 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

Related Examples

- “Invoke MATLAB Functions that Return Multiple Outputs” on page 2-7
- “Invoke MATLAB Functions that Return Zero Outputs” on page 2-5

Invoke MATLAB Functions that Return Multiple Outputs

In this section...

Receive the Results as Individual Variables

The connection between a Python client and a MATLAB Production Server instance is encapsulated in a `matlab.production_server.client.MWHttpClient` object. When you are expecting multiple return values from the server and want each return value saved in a variable, invoke MATLAB functions directly 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

Each variable is populated with a single return value.

For example, 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
```

Receive the Results as a Single Object

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

```
results = 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 variable is populated by a list containing all of the returned values.

For example, 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")
>>> copies = my_client.copier.copy("blue",10,nargout=2)
>>> print(copies)
["blue",10]
```

See Also

Related Examples

- “Invoke MATLAB Functions that Return a Single Output” on page 2-6
- “Invoke MATLAB Functions that Return Zero Outputs” on page 2-5

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()
```

Data Handling

Pass Data to MATLAB Production Server from Python

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 Arrays as Python Variables” on page 3-5)	Numeric array
`float`	`double`
`complex`	Complex `double`
`int`	`int32`(Windows®) `int64`(Linux® and Mac)
`long` [a]	`int64`
`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

a. `long` is a data type of Python 2.7 only

Handle Data Returned 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 Arrays as Python Variables” on page 3-5)
`double` `single`	`float`
Complex (any numeric type)	`complex`

`int8`
`uint8`
`int16`
`uint16`
`int32`
	`int`

`uint32`
`int64`
`uint64`
	`int`
	`long`

`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 (`table`, `containers.Map`, categorical array)	Not supported
Other object (e.g., Java®)	Not supported

MATLAB Output Argument Type (scalar unless otherwise noted)	Resulting Python Data Type
Function handle	Not supported
Sparse array	Not supported
String array	Not supported
Structure array	Not supported

MATLAB Arrays as Python Variables

In this section...

The `matlab` Python package provides array classes to represent arrays of MATLAB numeric types as Python variables. Other MATLAB types are also supported, as listed in "Pass Data to MATLAB from Python" (MATLAB).

Create MATLAB Arrays in Python

You can create MATLAB numeric arrays in a Python session by calling constructors from the `matlab` Python package (for example, `matlab.double`, `matlab.int32`). 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 the array with an optional `initializer` input argument that contains numbers. The `initializer` argument must be a Python sequence type such as a list or a tuple. The optional `size` input argument sets the size of the initialized array. To create multidimensional arrays, specify `initializer` to contain multiple sequences of numbers, or specify `size` to be multidimensional. You can create a MATLAB array of complex numbers by setting the optional `is_complex` keyword argument to `True`. The `mlarray` module provides the MATLAB array constructors listed in the table.

Class from `matlab` Package	Constructor Call in Python	MATLAB Numeric Type
`matlab.double`	`matlab.double( initializer=None, size=None, is_complex=False)`	Double precision

Class from matlab Package	Constructor Call in Python	MATLAB Numeric Type
matlab.single	matlab.single(initializer=None, size=None, is_complex=False)	Single precision
matlab.int8	matlab.int8(initializer=None, size=None, is_complex=False)	8-bit signed integer
matlab.int16	matlab.int16(initializer=None, size=None, is_complex=False)	16-bit signed integer
matlab.int32	matlab.int32(initializer=None, size=None, is_complex=False)	32-bit signed integer
matlab.int64[a]	matlab.int64(initializer=None, size=None, is_complex=False)	64-bit signed integer
matlab.uint8	matlab.uint8(initializer=None, size=None, is_complex=False)	8-bit unsigned integer
matlab.uint16	matlab.uint16(initializer=None, size=None, is_complex=False)	16-bit unsigned integer
matlab.uint32	matlab.uint32(initializer=None, size=None, is_complex=False)	32-bit unsigned integer
matlab.uint64[b]	matlab.uint64(initializer=None, size=None, is_complex=False)	64-bit unsigned integer

Class from matlab Package	Constructor Call in Python	MATLAB Numeric Type
`matlab.logical`	`matlab.logical( initializer=None, size=None)`[c]	Logical

a. In Python 2.7 on Windows, `matlab.int64` is converted to `int32` in MATLAB. Also, MATLAB cannot return an `int64` array to Python.
b. In Python 2.7 on Windows, `matlab.uint64` is converted to `uint32` in MATLAB. Also, MATLAB cannot return a `uint64` array to Python.
c. Logicals cannot be made into an array of complex numbers.

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)`.

MATLAB Array Attributes and Methods in Python

All MATLAB arrays created with `matlab` package constructors have the attributes and methods listed in the following table:

Attribute or Method	Purpose
`size`	Size of array returned as a tuple
`reshape(size)`	Reshape the array as specified by the sequence `size`

Multidimensional MATLAB Arrays in Python

In Python, you can create multidimensional MATLAB arrays of any numeric type. Use two Python lists 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]]
```

Note Slicing MATLAB arrays behaves differently from slicing a Python list. Slicing a MATLAB array returns a view instead of a shallow copy.

Given a MATLAB array and a Python list with the same values, assigning a slice results in different results.

```
>>>mlarray = matlab.int32([[1,2],[3,4],[5,6]])
>>>py_list = [[1,2],[3,4],[5,6]]
>>>mlarray[0] = mlarray[0][::-1]
>>>py_list[0] = py_list[0][::-1]
>>>mlarray[0]
matlab.int32([[2,2],[3,4],[5,6]])
>>>py_list
[[2,1],[3,4],[5,6]]
```

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.

```
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]]
```

See Also

Related Examples

- “Use MATLAB Arrays in Python”
- “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 Arrays as Python Variables”

API Reference

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])
```

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.

Methods

close	Close the connection to MATLAB Production Server instance, and release the resources used by the connection
<archive>.<func>	Call a MATLAB function from a Python client

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.

See Also

Topics

“Create a Default Connection” on page 2-3
“Configure the Connection Time Out” on page 2-4
“Invoke MATLAB Functions that Return a Single Output” on page 2-6
“Invoke MATLAB Functions that Return Multiple Outputs” on page 2-7

close

Class: matlab.production_server.client.MWHttpClient
Package: matlab.production_server

Close the connection to MATLAB Production Server instance, and release the resources used by the connection

Syntax

```
my_client.close()
```

Description

`my_client.close()` closes the connection between the client and the server instance. It also frees any resources being used by the connection.

Examples

Disconnect a Python Client from a Server Instance

```
import matlab
from production_server import client

my_client = client.MWHttpClient("http://mps_host:9190")
my_client.close()
```

See Also

<archive>.<func>

Class: matlab.production_server.client.MWHttpClient
Package: matlab.production_server

Call a MATLAB function from a Python client

Syntax

```
results = my_client.<archive>.<func>(in_args, nargout=nargs)
```

Description

`results = my_client.<archive>.<func>(in_args, nargout=nargs)` evaluates the specified function, *`func`*, hosted in the deployed archive, *`archive`*, and returns the results.

Input Arguments

`in_args` — input arguments to the function being evaluated
comma separated list of Python variables

Input arguments to the named MATLAB function, specified as a comma separated list of Python variables. The client API passes all input arguments to the MATLAB function.

`nargs` — number of output values expected from the server instance
1 (default)

Number of output arguments from the named MATLAB function, specified as an integer. Set to the number of output arguments when you call a MATLAB function that returns multiple arguments. The client processes up to the specified number of output arguments. If you specify 2 and the function returns 3 values, the client receives only the first two values.

Output Arguments

`results` — output values from the function
comma separated list of Python variables or a single list

One or more output arguments from a MATLAB function, returned as one or more Python variables or as a list.

Exceptions

`HTTPException`	Raised if there is a problem communicating with the server instance.
`MATLABException`	Raised if a function call fails to execute.

See Also

Topics
"Invoke MATLAB Functions that Return a Single Output" on page 2-6
"Invoke MATLAB Functions that Return Multiple Outputs" on page 2-7