system.util.sendRequestAsync

This feature is new in Ignition version **7.9.3**
Click here to check out the other new features

This function is used in **Python Scripting.**

Description

This function sends a message to the Gateway and expects a response. Works in a similar manner to the sendRequest function, except sendRequestAsync will send the request and then immediately return a handle for it. The Request handle has the following methods:

- get() - Block for result, throw an exception on failure.
- cancel() - Cancel the request. Any completion callback will be called with CancellationException
- block() - Like get(), but will return a Boolean True or False once complete, indicating completion success. If False, call getError() to get the exception object.
- getError() - Returns the error result or null. Similar to get(), in that this will block for a result.
- onSuccess(PyFunction) - Will set a function to run on a successful completion callback or set a new one if one was already defined in the sendRequestAsync call.
- onError(PyFunction) - Will set a function to run on a failed completion callback or set a new one if one was already defined in the sendRequestAsync call.

Client Permission Restrictions

This scripting function has no Client Permission restrictions.

Syntax

system.util.sendRequestAsync(project, messageHandler, payload, remoteServer, timeoutSec, onSuccess, onError)

- Parameters

  String project - The name of the project containing the message handler.

  String messageHandler - The name of the message handler that will fire upon receiving a message.

  PyDictionary payload - Optional. A PyDictionary which will get passed to the message handler. Use "payload" in the message handler to access dictionary variables.

  String hostName - Optional. Limits the message delivery to the client that has the specified network host name.

  String remoteServer - Optional. A string representing the target Gateway Server name. The message will be delivered to the remote Gateway over the Gateway Network. Upon delivery, the message is distributed to the local Gateway and clients as per the other parameters.

  String timeoutSec - Optional. The number of seconds before the sendRequest call times out.

  PyFunction onSuccess- Optional. Should take one argument, which will be the result from the message handler. Callback functions will be executed on the GUI thread, similar to system.util.invokeLater.

  PyFunction onError- Optional. Should take one argument, which will be the exception encountered. Callback functions will be executed on the GUI thread, similar to system.util.invokeLater .

- Returns

  Request Handle - The Request object that can be used while waiting for the message handler callback.

- Scope

  All

Parameter Order

This feature is new in Ignition version **7.9.6**
Click here to check out the other new features

As of 7.9.6, the order of parameters changed, and is reflected in the syntax list above. When calling the function by order, the arguments should be passed like so:

Current Parameter Order

```
system.util.sendRequestAsync(project, messageHandler, payload, hostName, remoteServer,
timeoutSec, onSuccess, onError)
```

In versions 7.9.3, 7.9.4, and 7.9.5 the **timeoutSec** parameter was at the end:

Legacy Parameter Order

```
system.util.sendRequestAsync(project, messageHandler, payload, hostName, remoteServer,
onSuccess, onError, timeoutSec)
```

In either case, passing parameters via keyword arguments can be used across all versions.

Code Examples

```
# This will call the message handler 'test', and will return a handle into myHandle.
# We then call get() on myHandle, which will block the script and will wait for a return or
throw an exception on failure.
myHandle = system.util.sendRequestAsync(project='ACME', messageHandler='test', payload=
{'number':55})
myHandle.get()
```

```
# This will call the message handler 'test', and will return a handle into myHandle.
# In this example, we will define a function to run when the message handler has successfully
finished, using the onSuccess function on the Request Handle.

# Note that function accepts a single argument for the message.
def successFunc(message):
        system.gui.messageBox("Successfully finished: %s" % message)


# We're specifying that the request should timeout after 5 seconds.
myHandle = system.util.sendRequestAsync(project='ACME', messageHandler='test', payload=
{'number':55}, timeoutSec=5)


# Call the Request Handler's onSuccess function, passing in successFunc.
myHandle.onSuccess(successFunc)
```