system.net.httpClient

This feature is new in Ignition version **8.0.6**
Click here to check out the other new features

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

Description
Provides a general use object that can be used to send and receive HTTP requests. The object created by this function is a wrapper around Java's HttpClient class. Usage requires creating a `JythonHttpClient` object with a call to `system.net.httpClient`, then calling a method (such as `get()`, `post()`) on the `JythonHttpClient` to actually issue a request.

Client Permission Restrictions
This scripting function has no Client Permission restrictions.

Syntax

system.net.httpClient()

- Parameters

  Integer timeout - A value, in milliseconds, to set the client's connect timeout setting to. [Optional. Defaults to 60000.]

  Boolean bypass_cert_validation - A boolean indicating whether the client should attempt to validate the certificates of remote servers, if connecting via HTTPS/SSL. [Optional. Defaults to True.]

  String username - A string indicating the username to use for authentication if the remote server requests authentication; specifically, by responding with a WWW-Authenticate or Proxy-Authenticate header. Only supports Basic authentication. If `username` is specified but not `password`, an empty string will be used for the password in the Basic Authentication response. [Optional. Defaults to None.]

  String password - A string indicating the password to use for authentication. [Optional. Defaults to None.]

  String proxy - The address of a proxy server, which will be used for HTTP and HTTPS traffic. If a port is not specified as part of that address, it will be assumed from the protocol in the URL, i.e. 80/443. [Optional. Defaults to None.]

  String cookie_policy - A string representing this client's cookie policy. Accepts values `"ACCEPT_ALL"`, `"ACCEPT_NONE"`, and `"ACCEPT_ORIGINAL_SERVER"` . [Defaults to `"ACCEPT_ORIGINAL_SERVER"`]

  String redirect_policy - A string representing this client's redirect policy. Acceptable values are listed below. [Optional. Defaults to "Normal".]

  - `"NEVER"` - never allow redirects
  - `"ALWAYS"` - allow redirects
  - `"NORMAL"` - allows redirects, except those that would downgrade to insecure addresses (i.e., HTTPS redirecting to HTTP)

  Callable customizer - A reference to a function. This function will be called with one argument (an instance of HttpClient.Builder). The function should operate on that builder instance, which allows for customization of the created HTTP client. [Optional. Defaults to None.]

- Returns

  JythonHttpClient - An object wrapped around an instance of Java's HttpClient class. The httpClient object has methods that can be called to execute HTTP requests against a server. See the panel below for more details.

- Scope

  Gateway, Vision Client, Perspective Session

Once a JythonHttpClient object has been created, it can be used to handle many HTTP requests without needing to create a new client. Individual HTTP requests can be made with the methods detailed below.

JythonHttpClient Methods

Methods

The following methods return either a Response object, or a Promise object that will eventually resolve to a Response object, if asynchronous. Asynchronous means that the method will be called, but will not block script execution - so multiple asynchronous calls to network services can be made in succession, without each call "waiting" for the result of the previous. Parameters for these functions are documented below.

Method	Description	Return type
`.get()`	Sends an HTTP GET call, blocking for a response.	Response
`.getAsync()`	Sends an HTTP GET call without blocking.	Promise
`.post()`	Sends an HTTP POST call, blocking for a response.	Response
`.postAsync()`	Sends an HTTP POST call without blocking.	Promise
`.put()`	Sends an HTTP PUT call, blocking for a response.	Response
`.putAsync()`	Sends an HTTP PUT call without blocking.	Promise
`.delete()`	Sends an HTTP DELETE call, blocking for a response.	Response
`.deleteAsync()`	Sends an HTTP DELETE call without blocking.	Promise
`.patch()`	Sends an HTTP PATCH call, blocking for a response.	Response
`.patchAsync()`	Sends an HTTP PATCH call without blocking.	Promise
`.head()`	Sends an HTTP HEAD call, blocking for a response.	Response
`.headAsync()`	Sends an HTTP HEAD call without blocking.	Promise
`.options()`	Sends an HTTP OPTIONS call, blocking for a response.	Response
`.optionsAsync()`	Sends an HTTP OPTIONS call without blocking.	Promise

`.trace()`	Sends an HTTP TRACE call, blocking for a response.	Response
`.traceAsync()`	Sends an HTTP TRACE call, without blocking for a response.	Promise
`.request()`	Sends an HTTP request, using a verb specified by the `method` parameter. Use this method in cases where a non-standard verb is required, and you need the call to block.	Response
`.requestAsync()`	Sends an HTTP request, with a verb specified by the `method` parameter. Use this method in cases where a non-standard verb is required, and you do not want the call to block.	Promise

Parameters

Parameters in this section can be used by any of the methods above. Exceptions to this rule will be defined on each parameter.

String url - The URL to connect to. [Required]

String method - The method to use in the request. [Required. Used by `.request()` and `.requestAsync()` only.]

String or Dictionary params - URL parameters to send with the request. [Optional. Defaults to None.]

- If supplied as a string, will be directly appended to the URL.
- If supplied as a dictionary, key/value pairs will be automatically URL encoded.

String or Dictionary or byte[] data - Data to send in the request. [Optional. Defaults to None.]

- String data will be sent with a Content-Type of "`text/plain; charset=UTF-8`", unless a different Content-Type header was specified.
- Dictionaries will be automatically encoded into JSON to send to the target server, with a Content-Type header set to "`application/json;charset=UTF-8`" unless a different Content-Type header was specified.
- Byte arrays will be streamed directly to the target server, with a Content-Type header of a `pplication/octet-stream` unless a different Content-Type header was specified.

String file - The path to a file, relative to the HTTP client instance. If specified, and the path is valid, the data in the file will be sent to the remote server. The file attribute overrides any value set in data; only the file's data will be sent. [Optional. Defaults to None.]

Dictionary headers - A dictionary of HTTP headers to send with the request. [Optional. Defaults to None.]

String username - Username to add to a Basic Authorization header in the outgoing request. If `username` is specified, but not `password`, the password is encoded as an empty string. [Optional. Defaults to None.]

String password - Password to add to a Basic Authorization header in the outgoing request. [Optional. Defaults to None.]

Integer timeout - The read timeout for this request, in milliseconds. [Optional. Defaults to 60000.]

JythonHttpClient Attributes

This section documents available attributes on the JythonHttpClient object.

Attribute	Description	Return Type
`.javaClient`	Returns the underlying Java HTTPClient instance.	HTTPClient
`.cookieManager`	Returns a CookieManager, which can be used to get or set cookies on requests from this client, or to override the cookie storage policy of the client.	CookieManager

CookieManager

Each JythonHttpClient instance has an attached CookieManager. This CookieManager can be accessed to retrieve cookies set by external web services, or to set cookies (ie, for authentication) before a request is made.

CookieManager Methods and Attributes

This section details methods on the CookieManager. Setting the cookie policy is easiest on the initial `system.net.httpClient` call, but the policy on the CookieManager can be overridden with a call to the built-in `setCookiePolicy` method. Policies are defined in the Java CookiePolicy interface.

Methods and Attributes

Method and/or Attribute	Description	Return type

`.getCookieStore()`
 	
`.cookieStore`
	Returns the underlying CookieStore, which can be used to add, remove, or get cookies that have been set by requests from the parent HttpClient instance. See the Java CookieStore interface for more information.	CookieStore

`.getCookieManager()`
 	
`.cookieManager`
	Sends an HTTP GET call, blocking for a response.	CookiePolicy

`.setCookiePolicy(policy)`	Sets a new CookiePolicy. See the Java CookiePolicy interface for more information.	None

```
from java.net import CookiePolicy

client = system.net.httpClient()
manager = client.getCookieManager()
manager.setCookiePolicy(CookiePolicy.ACCEPT_NONE)
```

Response Object

This section documents the Response object, returned by the request methods on the JythonHttpClient object. This object is simply a wrapper for Java's HTTPResponse object.

Response Methods and Attributes

This section details methods on the Response object.

Methods and Attributes

Method and/or Attribute	Description	Data type
`.getBody()` `.body`	Returns the response content directly.	Byte Array
`.getJson([encoding])` `.json`	Returns the response content as a dictionary, decoded with the encoding specified by the response. The optional encoding parameter can be used to specified how the JSON should be decoded before being mapped into Python objects (dictionary, list, etc). If the response is not valid JSON, an error will be thrown.	Dictionary
`.getText([encoding])` `.text`	Returns the response content, decoded as a string - either with the charset specified by the response (defaulting to UTF-8 if not specified by the remote server), or using the encoding specified in the function call.	String
`.getStatusCode()` `.statusCode`	Return the status code of the response object (i.e., 200 or 404).	Integer
`.isGood()` `.good`	Returns True if the response was good (i.e., 200) or False if it was a client or server error (status code between 400 and 599).	Boolean
`.isClientError()` `.clientError`	Returns True if the response was a client error, as in an HTTP 4XX response.	Boolean
`.isServerError()` `.serverError`	Returns True if the response was a server error, as in an HTTP 5XX response.	Boolean
`.getUrl()` `.url`	Returns the URL this Response connected to.	String

`.getHeaders()` `.headers`	Returns a case-insensitive “dictionary” of headers present on the response. Values will always be in a list, even if only a single header value was returned.	Dictionary
`.getJavaResponse()` `.javaResponse`	Returns the underlying Java HttpResponse behind this Response.	HttpResponse
`.getCookieManager()` `.cookieManager`	Returns the CookieManager. See the CookieManager section for more details.	CookieManager
`.getRequest()` `.request`	Returns a RequestWrapper object, which has details about the original request that was sent to return this response.	RequestWrapper

Promise Object

This section documents the Promise object, which is returned by the asynchronous methods available on the JythonHttpClient object. This object is a wrapper around Java's CompletableFuture `class`, and will return some different object once completed with `.get()`.

Promise Methods and Attributes

Method and/or Attribute	Description	Data type
`.get([timeout])`	Block for timeout until a result is available. The result object can technically be any type, if chaining, but will be a Response object when calling one of the HTTPClient methods. If the timeout is met without a result, an exception will be thrown. The timeout, if unspecified, is 60 seconds.	Any
`.then(callback)`	Allows for chaining, by returning a new Promise which wraps the provided callback. The callback parameter should be a Python function that either accepts two arguments (the result, or an error, either of which can be None) *or* a single argument, but is able to accept exceptions as well as valid values.	Promise
`.handleException(callback)`	In the event of an exception in a potential chain of promises, `handleException` will be called with one argument (the thrown error) and is expected to return a new fallback value for the next step in the promise chain.	Promise
`.whenComplete(callback)`	Call the provided callback when this promise finishes evaluating. Callback will be called with return value as the first argument, and any thrown error as the second argument. Any return value will be ignored.	None
`.cancel()`	Attempt to cancel the wrapped Java future. Returns True if the cancellation succeeded.	Boolean
`.getFuture()` `.future`	Returns the underlying Java CompletableFuture object that this Promise contains.	CompletableFuture
`.isDone()` `.done`	Returns True if the underlying future has completed - regardless of whether it was a good result or exception.	Boolean

RequestWrapper Object

This section documents the RequestWrapper object, which is simply a wrapper around Java's HTTPRequest object. This object can be used to determine details about the request that was originally sent to populate a Response object.

RequestWrapper Methods and Attributes

This section details methods on the RequestWrapper object.

Methods

Method and /or Attribute	Description	Data type
`.getUrl()` `.url`	Returns the actual URL that was contacted in the request.	String
`.getMethod()` `.method`	Return the HTTP method used in this request; GET, POST, PATCH, etc.	String
`.getHeaders()` `.headers`	Returns a case-insensitive “dictionary” of headers present on the request. Values will always be in a list, even if only a single value is present.	Dictionary
`.getTimeout()` `.timeout`	Returns the timeout this query was set to, or -1 if the timeout was invalid.	Integer
`.getVersion()` `.version`	Returns the HTTP version used for this request; will be either HTTP_1_1 or HTTP_2.	String
`.getJavaRequest()` `.javaRequest`	Returns the underlying Java HttpRequest object directly.	HttpRequest

Code Examples

Example

```
# Create the JythonHttpClient
client = system.net.httpClient()

# Sent a GET request
response = client.get("https://httpbin.org/get", params={"a": 1, "b": 2})

# Validate the response
if response.good:
        # Do something with the response
    print response.json['args']['a']
```

Example - Waiting for a Response

```
client = system.net.httpClient()

# Send a non-blocking request to an endpoint that will wait 3 seconds
promise = client.getAsync("https://httpbin.org/delay/3", params={"a": 1, "b": 2})

# This will print before we get a response from the endpoint.
print "doing something while waiting..."
# do more work here...

# After the work on the previous lines, we can now block and wait for a response
response = promise.get()
if response.good:
    print response.json['args']['a']
```

Keywords

system nav httpClient, nav.httpClient