The HTTP Module

Introduction

This blurb is about using QDL's HTTP module. This module allows you to do basic operations to a website using the HTTP protocol.

Loading the module

This is a Java module and is included in the standard distribution, but is not loaded, so to use it load it by issuing

```
  q := module_load('edu.uiuc.ncsa.qdl.extensions.http.QDLHTTPLoader','java')
 module_import(q)
http
```

or use jload which does it all in one fell swoop:

```
  jload(info().'lib'.'tools'.'http')
http
```

Supported functions

Name	Description	Comment
close()	close the connection	All requests will fail until open() is called
delete()	DELETE	Use current host
delete(parameters.)	DELETE	Use current host, add parameters
delete(uri_path, parameters.)	DELETE	Append uri_path to host, add parameters
get()	GET	Use current host
get(parameters.)	GET	Use host, add parameters. is a stem
get(uri_path, parameters.)	GET	Append path_uri to host, use parameters
headers()	list the current default headers	
headers(arg.)	set the default headers	
host()	get the current host	
host(host_name)	set the current host	
is_open()	is the connection open?	
is_json(response.)	Is the response content of type JSON?	This and is_text accept either the response or just the headers.

is_text(response.)	Is the response content of type text?	This does not include JSON since there is a separate method for that.
open()	open a new connection	
open(insecure)	open a new, insecure connection	insecure is a boolean, which if **true** will turn off security for SSL. Default is **false**.
post(arg | arg.)	POST	Payload may be a string or a stem.
post(uri_path, arg | arg.)	POST	Append uri_path to host, send payload
put(arg | arg.)	PUT	Payload may be a string or stem
put(uri_path, arg | arg.)	PUT	Append uri_path to host, send payload

Get returns a stem with entries for status, content and any returned headers.

Typical examples

Assuming you have loaded the above, open up a connection and get

```
    jload(info().'lib'.'tools'.'http')
http
    http#host('https://didact-patto.dev.umccr.org/api/visa') ;
    http#open();
true
    z. := http#get({'sub':'https://nagim.dev/p/wjaha-ppqrg-10000'});
    z.
{
 headers: {
  Connection:keep-alive,
  etag:W/"e6-suhkGbMm3fkbNhOR6bOIwIgkh8A",
  Apigw-Requestid:Gv9mdhv4SwMEMHw=,
  Content-Length:230,
  Date:Tue, 05 Oct 2021 19:37:04 GMT,
  Content-Type:application/json; charset=utf-8,
  X-Powered-By:Express
 },
 content: [
    {

s:XnKFkl4RTXtB2DDOf5f4yLtfcTaCGyqMxIV8Q42zX_XR1p9Cnxeqg2KI_4UCzcJZ2XGv_hlqVGOW5_3FE
9ZHCQ,
   v:c:8XZF4195109CIIERC35P577HAM et:1633549022 iu:https://nagim.dev/p/wjaha-ppqrg-
10000 iv:2f69e2650aed4f0e,
   k:rfc8032-7.1-test1
  }
  ],
 status: {
  code:200,
  message:OK
 }
}
```

```
   is_json(z.)
true
```

So we see the various components of the response, z.:

- `headers.` - A stem of the headers, where the key is the name of the header and the value is its value (as a string, so `Content-Length` is not a number).
- `content.` - The exact content. Here, it is an array with a single element and note that
  `headers.Content-Type`
  contains `application/json` and hence was in JSON format. If the content type is anything else, it will be returned as a stem of lines. The **`is_json`** function tell you if the content type was JSON. In this example, it was a JSON blob with 3 entries.
- `status.` - The http status, which includes the status code and the message from the server. Anything other than something in the 200 range is an error.

Note that there are other options for `content.` but it will always be an array. For instance, from other servers it may be the lines in the body of the response if the `Content-Type` is `form_encoding`. In that case, you will have to loop through the lines and process each of them in turn.