
webGet

Lua HTTP/HTTPS client with XML and XML/RDF parser

Copyright © 2010 StrixDB. Freely available under the terms of the [StrixDB license](#).

Table of Content

[Overview](#)

[Status](#)

[Download and Installation](#)

[History](#)

[Manual](#)

[Reference](#)

[Methods](#)

[Parser functions](#)

[Helper functions](#)

Overview

webGet is a regular lua 5.1 module that could be used to download file with HTTP/HTTPS and FTP requests.

webGet can use cookies and a download cache .

webGet provides also a simple [SAX](#) XML parser to process files during download. XML/RDF files are parsed and retrieved as triples.

webGet could submit data to web server with POST request.

The SAX parser could also be used to parse local files or Lua strings.

Status

Current version is 1.0. It was developed for Lua 5.1.
Current version is only available for Windows (use **WinInet**).

The current version is a beta release and could be used free of charges and available under the terms of the StrixDB license for beta releases.

webGet uses the Expat library and the **WinInet** API of Windows.

Download and installation

webGet can be downloaded as a Windows binary from
<http://www.strixDB.com/download.html>

It doesn't require extra modules or dependencies and need only LUA5.1.dll (the DLL provided by [LuaBinaries](#)). Windows System dependencies are KERNEL32.DLL and WININET.DLL

Installation is straightforward. Just use the require function of Lua.

```
C:\StrixDB\release>lua
Lua 5.1.4 Copyright (C) 1994-2008 Lua.org, PUC-Rio
> require('webGet')
>
```

History

- 22-07-2010 Version1.2 https compliant. <url> string could be replaced with {url=<string>,referer=<string>,types=<string>}
- 17-07-2010 Version1.1 module renamed from httpClient to webGet. Add of the functions **cli:GETtriples** and **cli:POSTtriples**.
- 14-07-2010 Initial release Version1.0

Manual

The above commented Lua code is self-explaining (and show the zen simplicity of the API) :

```
cli = webGet.new{}
local address = 'http://www.strixDB.com/samples/animals.rdf'

local rc,error=cli:GET(address, print) -- print the file
assert(rc==200)
rc,error=cli:GETXML(address, print) -- print the first tag, his
attribute(a table) and XML depth

function printAsNT(subject,predicate,object,datatype,lang)
    -- build printable subject
    if subject:byte(1)~=95 then
        subject='<..subject..'>'
    end
    -- build printable predicate
    predicate = '<..predicate..'>'
    -- build printable object
    if datatype==nil then
        object = '<..object..'>'
    else
        object = '"..object.."'
        if lang~=nil then
            object = object..'@'..lang
        end
        if datatype~='http://www.w3.org/2001/XMLSchema#string' then
            object = object..'^^'..datatype
        end
    end
    print(subject,predicate,obj, '.')
end

rc,error=cli:GETtriples(address, printAsNT)
cli:close()
```

Using https protocol is very simple. Certificates are shared with your current Microsoft Internet Explorer Browser.

If some **webGet** connection with https fails due to an invalid certificate (show the error message), just launch Internet Explore with the same URL and follow the wizard to accept (or reject) the certificate. An example of https connection:

```
rc,err = cli:GET(
{url='https://store.in.one.click/basket/',referer='http://store.in.one.click/android/killed\_iPod/'}, print )
```

NOTE: url in connection method could be a string or a table describing url, referer and allowed types.

Reference

webGet.new{[options]}

This function returns a **Wininet** Internet Session. By default, the session use download cache and the Internet Explorer proxy settings. The user agent is by default “**Mozilla/4.0**”.

Options are a table with the following optional entries :

- cache=true/false
- proxy=<string>
- proxyBypass=<string>
- userAgent=<string>

The Lua userdata returned by this function is used to make http requests.

webGet.setCookie(<url>,<name>, <value>)

This function associate to <url> (a lua string) a cookie named <name> with the given <value> (a lua string).

webGet.setCookie(<url>,<table>)

This function associate to <url> (a lua string) all the cookies specified by the table. Each entry of table is a cookie name.

webGet.getCookie(<url>)

This function returns a table of all cookies associated to the given <url>. Sample code printing cookies:

```
local cookies = webGet.getCookie('http://www.google.com')
table.foreach(cookies, print)
```

Internet Session Methods

The methods are available for an internet session :

```
cli = webGet.new { }
```

cli:close()

Closes the session, freeing all memory.

```
rc, str = cli:GET(<url>)  
rc, str = cli:GET(<url>, function f(txt) end)
```

The first method parameter **<url>** could be a Lua string (example: <http://www.strixDB.com>) or a table with the following entries:

- **url=<string>** required parameter.
- **referer=<string>**, the URL referer (see. HTML protocol definition).
- **types={<string> [,<string>]}** a table of all allowed types (for more information, see documentation of parameter ***lpIpszAcceptTypes** for the WinInet HttpOpenRequest Function)

For all the methods GET, POST, GETXML, POSTXML, GETtriples, POSTtriples, the first method parameter could be such a <string> or a Lua Table.

This method returns the http status code (examples: rc=200 if everything OK, rc=404 for file not found, etc...) and a string containing the file content OR an error message if rc is not 200.

The second form of this method call the function for each block of data retrieved from internet.

```
cli = webGet.new {}  
local rc, str = cli:GET('http://www.strixDB.com/2010/parentShip.rlog')  
assert(rc==200)  
-- now str contains the targeted file  
  
local out = io.open('samesRules.rlog', "w")  
local rc, str = cli:GET('http://www.strixDB.com/2010/parentShip.rlog')  
function(buffer) out:write(buffer) end )  
assert(rc==200)  
out:close()  
-- now the local file targeted file
```

```
rc, str = cli:GETXML(<url> [,<startFunc> [,<dataFunc> [,<endFunc>]]] )
```

The parameters functions are the classical SAX functions.

- The <startFunction> has 3 parameters : the tag name, a Lua table of tag attributes and the depth.
- The <dataFunction> has 1 parameter : the CDATA text.
- The <endFunction> has 1 parameter : the tag name.

The following example analyze a remote XML file :

```
cli = webGet.new {}

function startF(tag,attrs,depth)
    local x = tag
    for i,v in pairs(attrs) do
        x = x..' '..i..'='..'v
    end
    print(x)
end

function dataF(txt) print('txt=',txt) end
function endF(txt) print('end tag=',txt) end

local rc,str = cli:GETXML('http://someFile/',startF,dataF,endF)
assert(rc==200)
```

c,errmsg = cli:GETtriples(<url>, tripleCallback)

This function calls the lua function **tripleCallback** for each RDF triple found in the file at **<url>** address. The signature of the callback function is :

```
function (subject, predicate, object, datatype, lang)
```

where subject contains resource or blank node ID, predicate has a resource and object could be :

- a resource and in this case **datatype** and **lang** are nil
- a string and in this case **datatype**='http://www.w3.org/2001/XMLSchema#string'. lang is the language has defined RFC 4646.
- a literal value of type defined by **datatype**.

The [above example](#) shows such a call back function processing triples in N-Triple format.

rc,str = cli:POST(<url>, <dataToSend> [,function f(txt)end])

Function working as **cli:GET** . The provided data (a Lua string) are send in form encoding to the <url>.

cli:POSTXML(<url>,datas [,<startFunc> [,<dataFunc> [,<endFunc>]]])

Function working as **cli:GETXML**. The provided data (a Lua string) are send in form encoding to the <url>.

c,errmsg = cli:POSTtriples(<url>, <dataToSend> , tripleCallback)

Function working as **POST** but retrieving the RDF triples as with **GETtriples**.

cli:expandNS(true/false)

Flag forcing the XML parser to expand or not expand namespaces.

cli:base(<base>)

Method to set the base URI of the XML parser.

cli:parse(<xmlstring> [,<startFunc> [,<dataFunc> [,<endFunc>]]])

Parse the <xmlstring>.

cli:flag()

return a string describing the current **WinInet** session flags.

cli:flag(<flagName>, true/false)

Set the corresponding flag for the current **WinInet** session. See **WinInet** documentation for more information. <flagName> must be one of :

- “cache” => INTERNET_FLAG_DONT_CACHE
- “reload” => INTERNET_FLAG_RELOAD
- “secure” => INTERNET_FLAG_SECURE
- “redirect” => INTERNET_FLAG_NO_AUTO_REDIRECT

cli:user(<userName>,<password>)

Set the user name and password for the session.

Parser functions

webGet.parse(<xmlstring> [,<startFunc> [,<dataFunc> [,<endFunc>]]])

Parse the <xmlstring> without expanding namespaces. See method **cli:GETXML** for the other parameters.

webGet.parseNS(<xmlstring> [,<startFunc> [,<dataFunc> [,<endFunc>]]])

Parse the <xmlstring> expanding namespaces. See method **cli:GETXML** for the other parameters.

Helper functions

webGet.time()
webGet.time(<integer>)

This helper function returns a date in a string in the standard Web time format (W3C DTF).

Without argument, the current dateTime is returned. The second form translate the integer (supposed to be a C time_t integer).

```
> = webGet.time()  
2010-07-2010-07-14T10:22:16Z  
>
```

webGet.encode(<string>)

This function returns the URL encoding (a lua string) of the argument string.

webGet.checkUTF8(<path>)

This function scans a directory (<path>) or a file (<path>) and analyzes if the files are correctly encoded in UTF8 (with or without UTF8 BOM).

OK means a file with no unicode characters or UTF8 unicode characters.

BOM means that the specific UTF-8 BOM sequence was found.

This function could be helpfull to check directories of Lua scripts.

```
> webGet.checkUTF8('pub/')  
BOM      pub/index.html
```



```
BOM      pub/res\datalog.css
          pub/res\energy.gif
OK       pub/res\foafGraph.rdf
OK       pub/res\graphDefinitions.xml
          pub/res\hierar.gif
          pub/res\ortho.gif
          pub/res\persons\bart.gif
          pub/res\persons\brockman.gif
          .....
          pub/res\persons\troy_mcclure.gif
OK       pub/res\test.html
OK       pub/scripts\script1.lua
BOM      pub/scripts\test1.html
BOM      pub/sparql.html
OK       pub/testform.html
>
```