Experimenting with the REST API – What Tools can I use? – Part III

In the third part (and subsequent posts) of this blog series on the REST API, we are going to focus on a few of the common tools that enable you to interact with the REST API. This is by no means complete or comprehensive (and it not intended to be), but serves as an introduction about a few of the most commonly used UI and Command Line tools that I have seen and that you can also use to interact with the REST API.

- REST Client – cURL
- REST Client – Firefox RESTClient 2.0.3 Extension
- REST Client – Java RESTClient WizTools.org from VMware
- NSX for vSphere Networking and Security API Programming Guide
- vSphere Managed Object Browser
- XML Validator
    - http://www.w3schools.com/xml/xml_validator.asp
- xmllint(1) command-line utility
- Base 64 Encoding
    - http://ostermiller.org/calc/encode.html

I am going to start with cURL (as this is one my favourite tools) and what I mostly use. I just like the simplicity of this command line tool, though there are many options that can be used to, so you have the best of both worlds, nice hey!

cURL Client – Installing

1. cURL is installed by default on most current Operating Systems and is normally located in /usr/bin/curl
2. If cURL is not installed then obtain the curl executable source code (curl 7.40.0 was released on the 8th of January 2015) or curl binary. Note that the curl source code and binary releases are not always at the same release level. This depends on what options curl is compiled with at runtime.
3. Now, if you download the source code to build your own version of curl with your own options, you will download the current file which is ***curl-7.40.0.tar.gz.*** This file is a gzip tar file so do the following to extract the contents and follow this procedure (on performed on Mac OS X).
    1. Decompress and untar the curl file you downloaded
    ```
    $ tar -xvzf curl-7.40.0.tar.gz
    ```
    2. cd into the curl directory that was automatically created
    ```
    $ cd curl-7.40.0
    ```
    3. In a typical unix installation after you have unpacked the source code will be to configure (choose what options you require) followed by a make, then a make install to install the curl binary.
    ```
    $ ./configure
    ```
    4. When the configure process has completed successfully you will see the following:

```
curl version:     7.40.0
Host setup:       x86_64-apple-darwin14.0.0
Install prefix:   /usr/local
Compiler:         gcc
SSL support:      enabled (OpenSSL)
SSH support:      no      (--with-libssh2)
zlib support:     enabled
GSS-API support:  no      (--with-gssapi)
TLS-SRP support:  no      (--enable-tls-srp)
resolver:         default (--enable-ares / --enable-threaded-resolver)
IPv6 support:     enabled
Unix sockets support: enabled
IDN support:      no      (--with-{libidn,winidn})
Build libcurl:    Shared=yes, Static=yes
Built-in manual:  enabled
--libcurl option: enabled (--disable-libcurl-option)
Verbose errors:   enabled (--disable-verbose)
SSPI support:     no      (--enable-sspi)
ca cert bundle:   no
ca cert path:     no
LDAP support:     enabled (OpenLDAP)
LDAPS support:    enabled
RTSP support:     enabled
RTMP support:     no      (--with-librtmp)
metalink support: no      (--with-libmetalink)
HTTP2 support:    disabled (--with-nghttp2)
Protocols:        DICT FILE FTP FTPS GOPHER HTTP HTTPS IMAP IMAPS LDAP LDAPS POP3 POP3S RTSP SMB SMBS SMTP SMTPS TELNET TFTP
```

```
$ gcc -v
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 6.0 (clang-600.0.56) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin14.0.0
Thread model: posix
$
```

5. Now run make (of course you will need a compiler installed). I use gcc.
6. Run make
7.
```
$ make
```
8. Now run make install
```
$ sudo make install
Password:
```
9. The curl binary will now be installed as in the following example if all the above steps have completed successfully.
```
$ curl --version
curl 7.40.0 (x86_64-apple-darwin14.0.0) libcurl/7.40.0 OpenSSL/0.9.8z zlib/1.2.5
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtsp smb smbs smtp smtps telnet tftp
Features: IPv6 Largefile NTLM NTLM_WB SSL libz UnixSockets
$
```
10. Now we have the latest curl binary installed we can test it! What is the most basic test you can perform? Well, something like this.
```
$ curl -I www.vmware.com
```
11. This will return (if successful) the following. I can not be much more simpler than that hey! You are looking for **200 OK** which is as we discussed in the introduction to the REST API – Part II REST Request Methods and Responses, the return code of 200 is the return code the client hopes to see! It indicates that the **REST API** successfully carried out whatever action the client requested, and that no more specific code in the 2xx series is appropriate.
```
$ curl -I www.vmware.com
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=utf-8
Content-Language: en-GB
Date: Sat, 17 Jan 2015 14:48:03 GMT
Connection: keep-alive

$
```

Now that we have looked at the very basics of getting the latest version of curl(1) installed, let us now focus on interacting with the REST API using curl(1) to communicate with VMware's NSX for vSphere Networking and Security Platform.

So, where do we begin? First, we need to know what the required configuration options used by the curl(1) client are for us to communicate successfully with the NSX for vSphere platform. The following table shows the curl(1) options that will be used in the following examples.

OPTIONS	COMMENT
-u (lowercase)	This specifies the username
-k (lowercase)	(SSL) This option explicitly allows curl to perform "insecure" SSL connections and transfers.
-H (Uppercase)	(HTTP) Extra header to use when getting a web page.
-H (Uppercase)	Host
-H (Uppercase)	Authorization
-X (Uppercase)	(HTTP) Specifies a custom request method to use when communicating with the HTTP server. Example, GET, POST, PUT, DELETE…

Now that we know what the the basic options we are going to use are, we can now perform some basic testing. I consider the following REST API example below to be akin to a ping(8) test. I mean this is probably one if not the simplest REST API requests you can send to the NSX for vSphere Manager. I also suggest that whenever you are working with the REST API and NSX for vSphere platform, that this is one command you always start with, as this proves basic communication.

Also remember that **All** REST APIs require authentication using Basic HTTP authentication. Each request must be authenticated individually. User names must also be either the local NSX for vSphere users or vCenter Server users added to the NSX for vSphere Manager with the appropriate role and scope assignments.

Enumerating the supported versions of the REST API with NSX for vSphere

- **URL : /api/versions**
- **GET Method**

```
$ curl -k -u admin:<Enter Password Here> -H "Accept:application/xml" -H 'Content-Type:application/xml' -X GET https://<NSX-Manager-IP-ADDRESS>/api/versions
```

In the example above you are going to enter the password of the user "admin" and the IP Address of the NSX for vSphere Manager. Once you have correctly provided these two parameters you will see the following output :

Take a moment to reflect on what we just did! This is one if not one of the most basic REST API requests that we can send to the NSX for vSphere Manager. OK, can you read this easily? Of course you can not, so this raises a very important question when working with the REST API using the command-line, and that is how can we make the REST API Response more user-friendly? Well, this is where JSON comes into its own. JavaScript Object Notation (or JSON as its referred to), is an open and text-based data exchange format, that provides a standardized data exchange format, and one that can be read more easily by humans. JSON is defined in the Networking group of the IETF in the Information category as RFC 4627. Unlike XML (in my opinion), it is human-readable, platform independent, and has wide availability. Data formatted according to JSON is lightweight and can be parsed by JavaScript implementations with incredible ease, making it an ideal data exchange format for web based applications.

Anyway, I digress, what I wanted to say is that based on our example above it would be so useful to get the REST API Request data returned in the JSON format. However, there is an issue currently with the NSX for vSphere platform, and that is that JSON is **NOT** currently **SUPPORTED**. What will happen is if you try and return the REST API Request in the JSON data format, you will get a message informing you that "**No JSON object could be decoded**". It is not all bad news, as in the fact that there are numerous objects in the NSX for vSphere platform that can be returned in the JSON data format. Please do read this point carefully, the JSON data format is **NOT** officially **SUPPORTED** in the current release of the NSX for vSphere Platform (Version: 6.1.2 Build 2318232).

So, this being the case how can I return the REST API Response in a more human readable way? Well, there are a few options available to you. When using the command-line I use a utility which is part of Mac OS X called xmllint(1) and if you want to use a UI option, then something like the Firefox RESTClient 2.0.3 Extension will work admirably for you. In the next Blog posts we will cover both xmllint(1) and the Firefox RESTClient.

The cURL source code and binaries are available from the following location:

- http://curl.haxx.se/download.html