Programmatic scripting with WSMAN

This Dell Technical White Paper provides information about programming scripts with WSMAN.

Author(s)

Thirumalaa Srinivas
Jon Hass

This white paper is for informational purposes only, and may contain typographical errors and technical inaccuracies. The content is provided as is, without express or implied warranties of any kind.

This document is for informational purposes only and may contain typographical errors and technical inaccuracies. The content is provided as is, without express or implied warranties of any kind.

April 2012| Rev 1.0

Contents

Executive Summary

This white paper is for systems administrators looking to program scripts with WSMAN clients and to harness the power of the secure and standards-based WSMAN service.

Introduction

Dell PowerEdge servers equipped with Integrated Dell Remote Access Controller (iDRAC) provide secure, simple, scriptable and standards-based remote management capability through Web Services for Management (WSMAN). WSMAN is a management transport protocol that enables a user to access systems management data objects and methods supported by the target platform. You can utilize the WSMAN interface by scripting WSMAN using command-line tools such as *winrm* on Windows and *wsmancli* on Linux. WSMAN can also be accessed from scripting languages like Python, which can run on both Windows and Linux. You may need to read specification documents to understand the terminology and concept in this document. If you are a systems administrator that typically works with command-line tools and scripts, then this white paper should benefit you.

This document helps you:

(A) Get started on programmatically scripting with WSMAN using Python.

(B) Learn common APIs to leverage in your custom scripts.

WSMAN clients

There are two primary WSMAN clients described in this document:

- Windows Remote Management (WinRM) CLI for Windows
- OpenWSMAN CLI for Linux

Both these WSMAN clients are equivalent in most of their feature set. They allow usage of most of the high frequency and high usage WSMAN operations. Reference the WinRM and OpenWSMAN CLI sections for sample output of the command-line tool’s help menu.

You can find more information on installation and usage of these CLI tools in the Where to Find More Information section. The section contains links to official reference guides from the CLI tool providers.

WinRM CLI

Sample output of WinRM CLI invocation.

```
C:\>winrm
Windows Remote Management Command Line Tool
Windows Remote Management (WinRM) is the Microsoft implementation of
the WS-Management protocol which provides a secure way to communicate
with local and remote computers using web services.

Usage:
  winrm OPERATION RESOURCE_URI [-SWITCH:VALUE [-SWITCH:VALUE] ...]
        [@{KEY=VALUE[;KEY=VALUE]...}]

For help on a specific operation:
  winrm g[et] -?        Retrieving management information.
  winrm s[et] -?        Modifying management information.
  winrm c[reate] -?     Creating new instances of management resources.
  winrm d[elete] -?     Remove an instance of a management resource.
  winrm e[numerate] -?  List all instances of a management resource.
  winrm i[nvoke] -?     Executes a method on a management resource.
  winrm id[entify] -?   Determines if a WS-Management implementation is
                        running on the remote machine.
  winrm quickconfig -?  Configures this machine to accept WS-Management
                        requests from other machines.
  winrm configSDDL -?   Modify an existing security descriptor for a URI.
  winrm helpmsg -?      Displays error message for the error code.

For help on related topics:
  winrm help uris       How to construct resource URIs.
  winrm help aliases    Abbreviations for URIs.
  winrm help config     Configuring WinRM client and service settings.
  winrm help certmapping Configuring client certificate access.
  winrm help remoting   How to access remote machines.
  winrm help auth       Providing credentials for remote access.
  winrm help input      Providing input to create, set, and invoke.
  winrm help switches   Other switches such as formatting, options, etc.
  winrm help proxy      Providing proxy information.
```

OpenWSMAN CLI

Sample output of OpenWSMAN CLI invocation.

```
[user@hostname ~]# wsman -?
Usage:
  wsman [Option...] <action> <Resource Uri>

Help Options
  -?, --help
  --help-all                                      Show help options
  --help-enumeration                              Enumeration Options
  --help-tests                                    Test Cases
  --help-cim                                      CIM Options
  --help-flags                                    Request Flags
  --help-event                                    Subscription Options

Application Options
  -d, --debug=1-6                                 Set the verbosity of debugging
output.
  -j, --encoding                                  Set request message encoding
  -c, --cacert=<filename>                         Certificate file to verify the
peer
  -A, --cert=<filename>                           Certificate file. The certificate
must be in PEM format.
  -K, --sslkey=<key>                              SSL Key.
  -u, --username=<username>                       User name
  -g, --path=<path>                               Service Path (default: 'wsman')
  -J, --input=<filename>                          File with resource for Create and
Put operations in XML, can be a SOAP envelope
  -p, --password=<password>                       User Password
  -h, --hostname=<hostname>                       Host name
  -b, --endpoint=<url>                            End point
  -P, --port=<port>                               Server Port
  -X, --proxy=<proxy>                             Proxy name
  -Y, --proxyauth=<proxyauth>                     Proxy user:pwd
  -y, --auth=<basic|digest|gss>                   Authentication Method
  -a, --method=<custom method>                    Method (Works only with 'invoke')
  -k, --prop=<key=val>                            Properties with key value pairs
(For 'put', 'invoke' and 'create')
  -C, --config-file=<file>                        Alternate configuration file
  -O, --out-file=<file>                           Write output to file
  -V, --noverifypeer                              Not to verify peer certificate
  -v, --noverifyhost                              Not to verify hostname
  -I, --transport-timeout=<time in sec>           Transport timeout in seconds
```

Scripting WSMAN clients using Python

1. Use Python version 2.7. Make sure this version is installed on your system and take care of version conflicts, if any. If you need help, refer to the python release site. We recommend this version of Python to leverage many of the list and xml tree iteration features that you can use in processing WSMAN’s CIM XML output format.

2. If you need references to implemented WSMAN scripts, refer to the python script packages from Dell Tech Center.

Pictorial View of the Environment

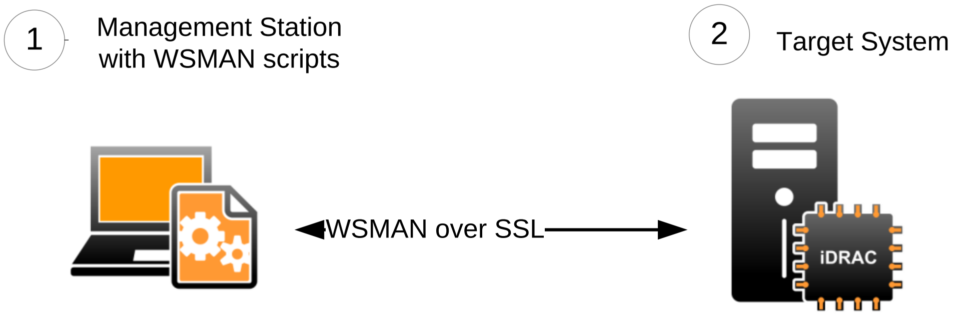


Figure 1 - Environment Diagram

Figure 1 shows the pictorial view of the environment. It starts with administrator (1) using the management station to run scripts to send WSMAN commands through an SSL connection. The target system in this case (2) is a Dell PowerEdge 11G or later server equipped with the iDRAC service processor.

The scripts can be interactive and menu-driven, or non-interactive with positional or option-based arguments. At Dell, we used either or both paradigms, based on the workflow or task that needed to be accomplished. We also took into consideration natural interaction patterns to provide a fluid user experience.

Common APIs

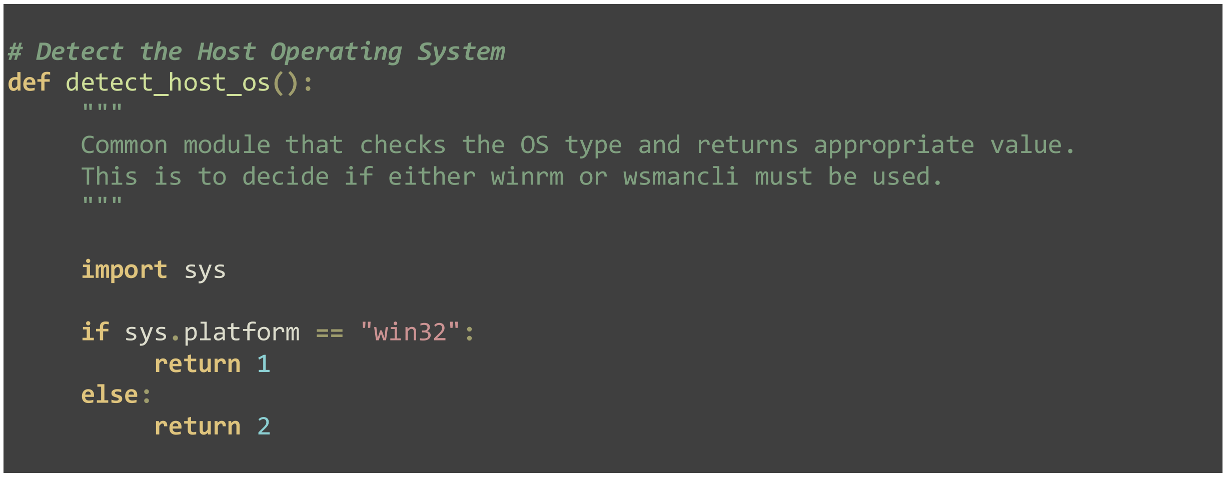
During our experience building scripts with WSMAN using Python, we found common patterns being used frequently. We saw the need to reuse the code and create common APIs from those frequent patterns. Some of them have been chosen for this white paper.

1. Detect Host Operating System
2. Construct WSMAN CLI command
3. Launch WSMAN CLI command
4. Ping test
5. Extract SSL certificate

In the following sections, this white paper explains these APIs and details their use. These APIs are used in the common files that come with the script packages posted at the Dell Tech Center. The link is available in the Where to Find More Information section. All the API examples are written using the Python programming language.

Detect Host Operating System

The host operating system must be detected as a pre-step to constructing the applicable WSMAN CLI command.



```
# Detect the Host Operating System
def detect_host_os():
     """
     Common module that checks the OS type and returns appropriate value.
     This is to decide if either winrm or wsmancli must be used.
     """

     import sys

     if sys.platform == "win32":
          return 1
     else:
          return 2
```

The API returns 1 for Windows or 2 for Linux, depending on the host operating system.
If the host OS is Windows, WinRM CLI commands need to be constructed.
If the host OS is Linux, OpenWSMAN CLI commands need to be constructed.

Construct WSMAN CLI commands

The following are the five WSMan operations that are most frequently used.

- Enumerate
- Enumerate keys
- Get
- Set
- Invoke

Having an API for each of these operations is important to enable custom WSMAN scripts creation. These APIs must detect the host operating system in order to construct the relevant WSMAN CLI command.

Enumerate

This API detects the host operating system and constructs the relevant WSMAN enumerate CLI command, using the input parameters provided. The input parameters for the API are classname, namespace, username, password, ipaddress and the SSL certificate. The certificate is defaulted to dummy.

The constructed command can be directly launched on the command line or shell of the host OS, using the API detailed in the Launch WSMAN CLI command section.

```
# Enumerate
def wsman_enumerate_command(classname, namespace, username, password, ipaddress,
cert = "dummy"):
    """
    Constructs the WSMan Enumerate Operation command based on the OS and returns
    the command to be used.
    """

    if detect_host_os() == 1:
        wsman_command = 'winrm e "http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s?__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8 -
a:basic -SkipCNcheck -SkipCAcheck -format:pretty'
        wsman_command = wsman_command % (classname, namespace, username, password,
ipaddress)
    else:
        wsman_command = 'wsman enumerate http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s -N %s -u %s -p %s -h %s -P 443 -v -j utf-8 -y basic -o -m 256 -c %s -
V'
        wsman_command = wsman_command % (classname, namespace, username, password,
ipaddress, cert)

    return wsman_command
```

Enumerate Keys

This API detects the host operating system and constructs the relevant WSMAN enumerate keys CLI command, using the input parameters provided. The input parameters for the API are classname, namespace, username, password, ipaddress and the SSL certificate. The certificate is defaulted to dummy.

The constructed command can be directly launched on the command line or shell of the host OS, using the API detailed in the Launch WSMAN CLI command section.

```
# Enumerate End Point Reference (EPR)
def wsman_enumerate_epr_command(classname, namespace, username, password,
ipaddress, cert = "dummy"):
    """
    Constructs the WSMan Enumerate EPR Operation command based on the OS and
returns
    the command to be used.
    """
    if detect_host_os() == 1:
        wsman_command = 'winrm e "http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s?__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8 -
a:basic -SkipCNcheck -SkipCAcheck -format:pretty -returntype:epr'
        wsman_command = wsman_command % (classname, namespace, username, password,
ipaddress)
    else:
        wsman_command = 'wsman enumerate http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s -N %s -u %s -p %s -h %s -P 443 -v -j utf-8 -y basic -o -m 256 -c %s -V
-M epr'
        wsman_command = wsman_command % (classname, namespace, username, password,
ipaddress, cert)

    return wsman_command
```

Get

This API detects the host operating system and constructs the relevant WSMAN get CLI command, using the input parameters provided. The input parameters for the API are classname, namespace, keys of the class instance in python dictionary format, username, password, ipaddress and the SSL certificate. The certificate is defaulted to dummy.

The constructed command can be directly launched on the command line or shell of the host OS, using the API detailed in the Launch WSMAN CLI command section.

```
# Get
def wsman_get_command(classname, namespace, key_dict, username, password,
ipaddress, cert = 'dummy'):
    """
    Constructs the WSMan Get Operation command based on the OS and returns
    the command to be used.
    Sample arguments:
       key_dict = {‘classname’:’DCIM_CS’,’name’:’srv:system’}
    """

    key_str = construct_key_str(key_dict)

    if detect_host_os() == 1:
        wsman_command = 'winrm g "http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/%s?%s+__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8
-a:basic -SkipCNcheck -SkipCAcheck -format:pretty'
        wsman_command = wsman_command % (classname, key_str, namespace, username,
password, ipaddress)
    else:
        wsman_command = 'wsman get http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/%s?%s -N %s -u %s -p %s -h %s -P 443 -v -j utf-8 -y basic -o -m 256 -c %s
-V'
        wsman_command = wsman_command % (classname, key_str, namespace, username,
password, ipaddress, cert)

    return wsman_command
```

Set

This API detects the host operating system and constructs the relevant WSMAN set CLI command, using the input parameters provided. The input parameters for the API are classname, namespace, keys of the class instance in python dictionary format, name-value pairs to be set on the instance in python dictionary format, username, password, ipaddress and the SSL certificate. The certificate is defaulted to dummy.

The constructed command can be directly launched on the command line or shell of the host OS, using the API detailed in the Launch WSMAN CLI command section.

```
# Set
def wsman_set_command(classname, namespace, key_dict, arg_dict, username,
password, ipaddress, cert = 'dummy'):
    """
    Constructs the WSMan Set Operation command based on the OS and returns
    the command to be used.
    Sample arguments:
       arg_dict = {‘classname’:’DCIM_CS’,’name’:’srv:system’}
    """

    # Constuct the keys from the input dictionary, into CLI-specific format
    key_str = construct_key_str(key_dict)
    # Construct the arguments from the input dictionary, into CLI-specific format
    arg_str = construct_arg_str(arg_dict)

    if detect_host_os() == 1:
        wsman_command = 'winrm s "http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s?%s+__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8
-a:basic -SkipCNcheck -SkipCAcheck -format:pretty @{%s}'
        wsman_command = wsman_command % (classname, key_str, namespace, username,
password, ipaddress, arg_str)
    else:
        wsman_command = 'wsman set http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/%s?%s -N %s -u %s -p %s -h %s -P 443 -v -j utf-8 -y basic -R -o -m 256 -c
%s -V %s'
        wsman_command = wsman_command % (classname, key_str, namespace, username,
password, ipaddress, arg_str, cert)
```

```
    return wsman_command
```

Invoke

This API detects the host operating system and constructs the relevant WSMAN invoke CLI command, using the input parameters provided. The input parameters for the API are classname, namespace, methodname, keys of the class instance in python dictionary format, name-value pairs to be set on the instance in python dictionary format, username, password, ipaddress, the SSL certificate and a filename if name-value pairs are passed through a file. The certificate is defaulted to dummy.

The constructed command can be directly launched on the command line or shell of the host OS, using the API detailed in the Launch WSMAN CLI command section.

```
# Invoke
def wsman_invoke_command(classname, namespace, methodname, key_dict, arg_dict,
username, password, ipaddress, cert = "dummy",filename=None):
    """
    Constructs the WSMan Invoke Operation command based on the OS and returns
    the command to be used.
    """

    # Constuct the keys from the input dictionary, into CLI-specific format
    key_str = construct_key_str(key_dict)
    if filename == None:
        # Construct the arguments from the input dictionary, into CLI-specific
format
        arg_str = construct_arg_str(arg_dict)

    if detect_host_os() == 1:
        # Check if filename is passed
        if filename == None:
            wsman_command = 'winrm i %s "http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/%s?%s+__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8
-a:basic -SkipCNcheck -SkipCAcheck -format:pretty @{%s}' % (methodname, classname,
key_str, namespace, username, password, ipaddress, arg_str)
        else:
            wsman_command = 'winrm i %s "http://schemas.dell.com/wbem/wscim/1/cim-
schema/2/%s?%s+__cimnamespace=%s" -u:%s -p:%s -r:https://%s/wsman -encoding:utf-8
-a:basic -SkipCNcheck -SkipCAcheck -format:pretty -file:%s' % (methodname,
classname, key_str, namespace, username, password, ipaddress, filename)
    else:
        # Check if filename is passed
        if filename == None:
            wsman_command = 'wsman invoke -a "%s"
http://schemas.dell.com/wbem/wscim/1/cim-schema/2/%s?%s -N %s -u %s -p %s -h %s -P
443 -v -j utf-8 -y basic -o -m 256 -c %s -V %s' % (methodname, classname, key_str,
namespace, username, password, ipaddress, cert, arg_str)
```

```
        else:
            wsman_command = 'wsman invoke -a "%s"
http://schemas.dell.com/wbem/wscim/1/cim-schema/2/%s?%s -N %s -u %s -p %s -h %s -P
443 -v -j utf-8 -y basic -o -m 256 -c %s -V -J "%s"'
            wsman_command = wsman_command % (methodname, classname, key_str,
namespace, username, password, ipaddress, cert, filename)

    return wsman_command
```

Launch WSMAN CLI command

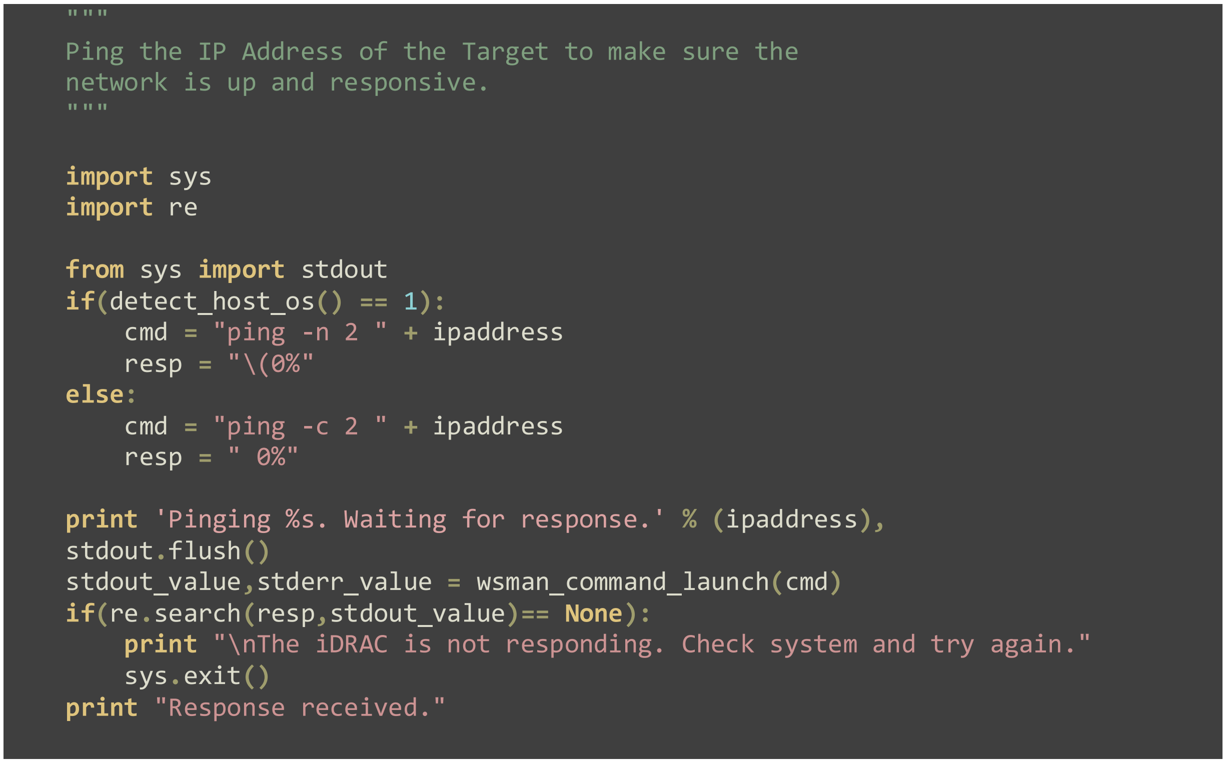
After the relevant WSMAN CLI command is constructed, it must be launched on the command line or shell. This must be done to communicate with the remote WSMAN service. The subprocess python module is used to achieve this. The response from the remote WSMAN service is stored in two strings stdout and stderr. After returned from the API, stdout must be checked to use the output data. If it is empty, then an error occurred. The stderr string must be used to print out the error.

```
# Launch the constructed WSMAN CLI command
def wsman_command_launch(wsman_command):
    """
    Executes the WSMan command on the shell (OS agnostic) and
    returns the standard out and error values.
    """

    import subprocess

    proc = subprocess.Popen(wsman_command,
                            shell=True,
                            stdin=subprocess.PIPE,
                            stdout=subprocess.PIPE,
                            stderr=subprocess.PIPE)
    stdout_value,stderr_value = proc.communicate()

    return stdout_value,stderr_value
```

Ping test

The ping API is used to test the network connectivity with the remote controller or device. This is a useful pre-step before launching WSMAN CLI commands to communicate with the remote WSMAN service. The implementation below detects the host OS and conducts the ping test. Based on the connectivity, the appropriate message will be displayed.

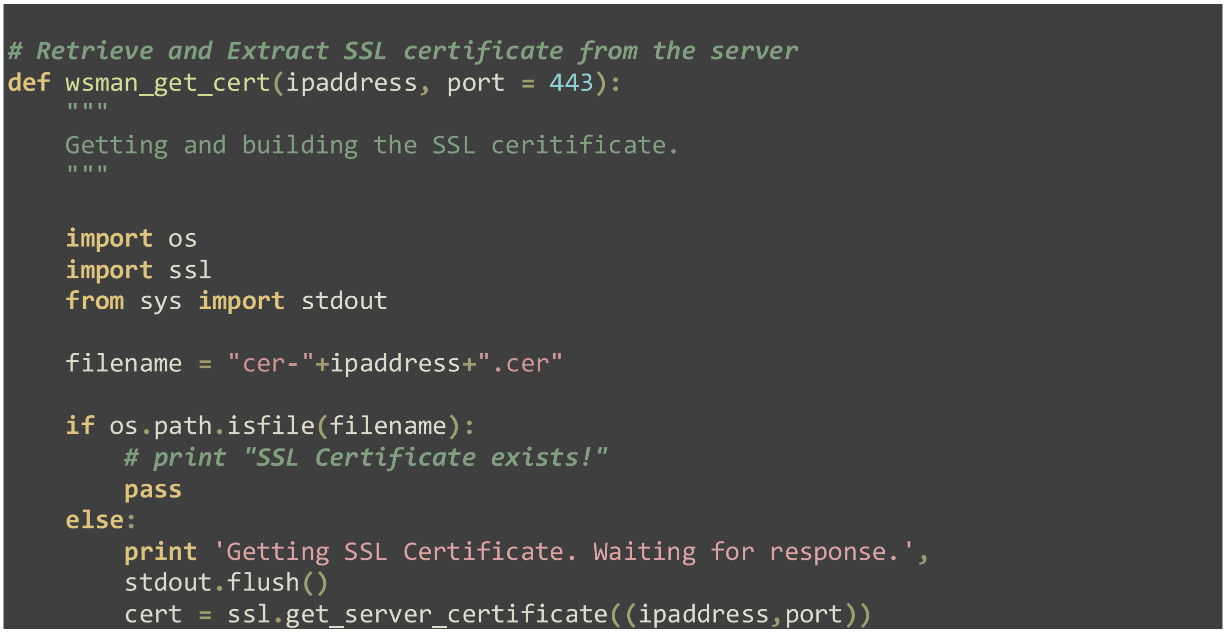
```
# Ping test
def ping(ipaddress):
```



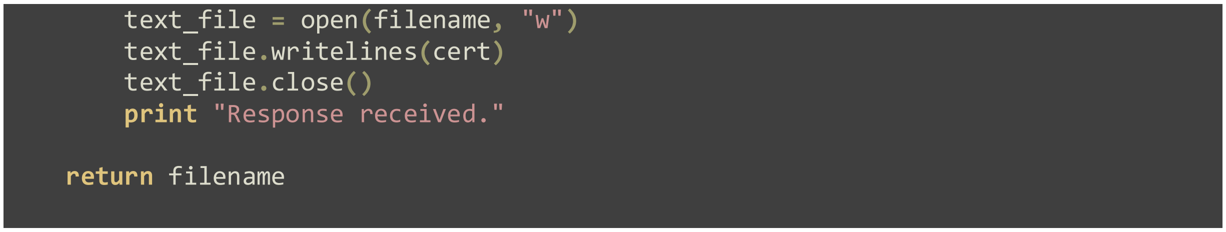
```
    """
    Ping the IP Address of the Target to make sure the
    network is up and responsive.
    """

    import sys
    import re

    from sys import stdout
    if(detect_host_os() == 1):
        cmd = "ping -n 2 " + ipaddress
        resp = "\(0%"
    else:
        cmd = "ping -c 2 " + ipaddress
        resp = " 0%"

    print 'Pinging %s. Waiting for response.' % (ipaddress),
    stdout.flush()
    stdout_value,stderr_value = wsman_command_launch(cmd)
    if(re.search(resp,stdout_value)== None):
        print "\nThe iDRAC is not responding. Check system and try again."
        sys.exit()
    print "Response received."
```

Extract SSL certificate

This API is used to check for presence of the SSL certificate in the file system. If the certificate is not present, it will be retrieved from the server and a new certificate file will be created.



```
# Retrieve and Extract SSL certificate from the server
def wsman_get_cert(ipaddress, port = 443):
    """
    Getting and building the SSL ceritificate.
    """

    import os
    import ssl
    from sys import stdout

    filename = "cer-"+ipaddress+".cer"

    if os.path.isfile(filename):
        # print "SSL Certificate exists!"
        pass
    else:
        print 'Getting SSL Certificate. Waiting for response.',
        stdout.flush()
        cert = ssl.get_server_certificate((ipaddress,port))
```



```
        text_file = open(filename, "w")
        text_file.writelines(cert)
        text_file.close()
        print "Response received."

    return filename
```

Important Notes

- Make sure your environment is configured well with WSMAN CLI tools, along with the appropriate version of Python.

- The target system that the scripts communicate with need to have WSMAN service running in the software stack. Dell's iDRACs are configured with WSMAN service in the firmware image.

Where to Find More Information

WSMAN Interface Guide for Linux:
http://en.community.dell.com/techcenter/extras/m/white_papers/20066176.aspx

WSMAN Interface Guide for Windows:
http://en.community.dell.com/techcenter/extras/m/white_papers/20066174.aspx

WSMAN command line open source for Linux (Openwsman):
http://sourceforge.net/projects/openwsman/

OpenWSMan installation instructions:
http://en.community.dell.com/techcenter/systems-management/w/wiki/3567.instructions-installing-openwsman-cli-on-linux.aspx

WSMAN command line for Windows (Winrm):
http://msdn.microsoft.com/en-us/library/windows/desktop/aa384291(v=VS.85).aspx

WSMAN scripts for the Dell Lifecycle Controller:
http://en.community.dell.com/techcenter/systems-management/w/wiki/scripting-the-dell-lifecycle-controller.aspx

General information for Lifecycle Controller – Remote Services

www.delltechcenter.com/lc

Summary

Using the tools readily available to Windows and with some work in Linux, this whitepaper provides information on how to (A) get started on programmatically scripting with WSMAN using Python and (B) create a set of functions or APIs for common operations to leverage in creating custom server management scripts. The ability to create custom scripts for remote and secure systems management enables you to be more productive and efficient.

Learn more

Visit Dell.com/PowerEdge for more information on Dell's enterprise-class servers.