

# Hands on Lab

## Script development on Dell EMC switches

Jose Gonzalez

May 5<sup>th</sup>, 2017

### Introduction

This is a quick tutorial to show you an environment to implement and debug automation scripts executed on Dell EMC switches running OS9 and OS10 Enterprise.

### Environment

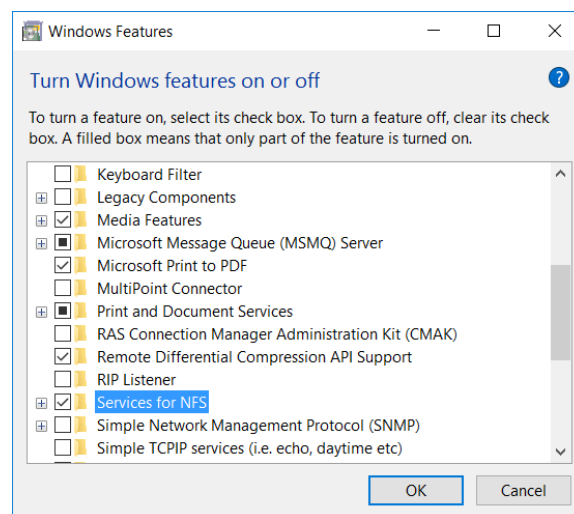
You will be using a PC with Windows to run a development IDE (Integrated Development Environment) and Dell EMC switches. Your PC must be able to reach the management interface of the Dell EMC switches.

You will also need an NFS share to access the scripts from a central location. OS9 and Windows 7/8/10 only support client NFS. For a NFS server, you can install any Linux distro on a VM or use OS10. There are free programs that offer server NFS functionality on Windows:

- <http://freenfs.sourceforge.net>
- <https://sourceforge.net/projects/winffsd>

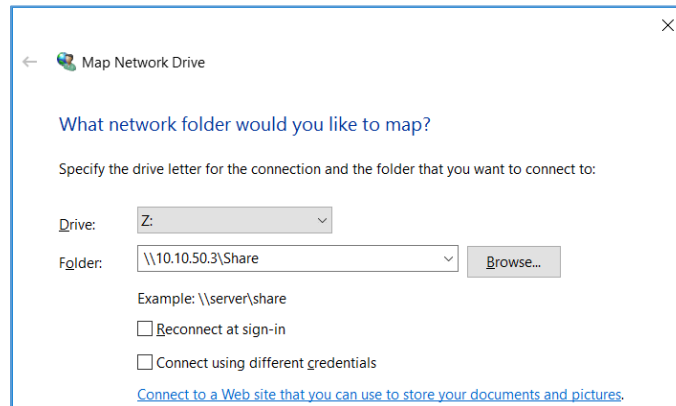
### Mount NFS share from Windows

Enable the NFS Service from the “Windows Features”



Then, open Windows Explorer, right-click “This PC”, and select “Map network drive”.

On that dialog box, enter the remote path with the format: \\<ip-or-hostname>:<share-path>



## Mount NFS share from OS9

Open a CLI session and enter the “mount” command

```
Dell# configure
Dell(conf)# mount nfs 10.10.50.3:/Share share
Dell(conf)# end
Dell# dir nfsmount://share
Directory of nfsmount://share

 1  drwx          64   May 06 2016 03:23:45 +00:00 .
 2  drwx          20   May 03 2017 12:32:11 +00:00 ..
...
Dell# script exec mount
...
10.10.50.3:/Share on /f10/mnt/nfs/share type nfs
```

## Mount NFS share from OS10

OS10 does not ship with support for NFS. You will need to install one of the following packages:

- nfs-common: NFS client. Use this option if you have already an NFS share somewhere else
- nfs-kernel-server: NFS client and server. Use this option to export directories from OS10

### NFS client

From the CLI session, enter the underlying Linux shell

```
OS10# system bash
admin@OS10:/config/home/admin$ sudo su
<Root credentials>
root@OS10:~# apt install nfs-common
root@OS10:~# mkdir /mnt/share
root@OS10:~# mount -t nfs 10.10.50.3:/Share /mnt/share
root@OS10:~# mount
...
10.10.50.3:/Share on /f10/mnt/nfs/share type nfs (rw,vers=3 ...)
```

NFS server

The file /etc/exports contains the definition of the subnets that are allowed to access the exported directories.

```

root@OS10:~# apt install nfs-kernel-server
root@OS10:~# mkdir /share
root@OS10:~# chmod a+rwX /share
root@OS10:~# vi /etc/exports
<Append at the end similar lines, one for each allowed subnet>
/share 10.10.0.0/16(rw,sync,no_subtree_check,insecure)

root@OS10:~# service nfs-kernel-server restart
root@OS10:~# showmount -e 127.0.0.1
Export list for 127.0.0.1:
/share 10.10.0.0/16

```

## Basic development in OS9

From the PC, create a new file on the Network drive using Linux line terminations:

**test.tcl**

```

#!/usr/bin/tclsh

set count 5
puts "Start"
for {set i 1} {$i <= $count} {incr i} {
    puts "TCL $i"
}
puts "End"

```

From OS9, verify you see the file:

```

Dell# dir nfsmount://share/test.tcl
 1  -rwx          112   May 2 2017 01:31:47 +00:00 test.tcl

Dell# show file nfsmount://share/test.tcl
#!/usr/bin/tclsh
...

```

If the file is not executable or not writable, change the permissions:

```

Dell# script exec chmod args "777 /f10/mnt/nfs/share/test.tcl"

```

Run it:

```
Dell# script exec /f10/mnt/nfs/share/test.tcl
Start
  TCL 1
  TCL 2
  TCL 3
  TCL 4
  TCL 5
End
```

By now, you are able to edit the script from the PC using any good visual editor and execute it from the switch. For small scripts, that is all you need. When you are satisfied with the results, you can copy the script to the switch flash:

```
Dell# script get nfsmount://share/test.tcl
!
112 bytes successfully copied
Dell# script exec test.tcl
Start
  TCL 1
  TCL 2
  TCL 3
  TCL 4
  TCL 5
End
```

Finally, remove the NFS share:

```
Dell(conf)# no mount nfs 10.10.50.3:/Share share
Dell(conf)# end
```

## Basic development in OS10

From the PC, create a new file on the Network drive using Linux line terminations:

### test.py

```
#!/usr/bin/python
if __name__=="__main__":
    count = 5
    i = 1
    print "Start"
    while i <= count:
        print " Python",i
        i = i + 1

    print "End"
```

```

OS10# system /share/test.py
Start
  Python 1
  Python 2
  Python 3
  Python 4
  Python 5
End

```

## Development and debugging with an IDE

For more complex scripts, you need an IDE with an integrated debugger and a language that supports remote debugging. Fortunately, OS9 and OS10 include several scripting options. Let's review them:

### OS9 (Except S3100)

Language	Where	Path to binary	Remote debugging?
Unix/Linux shells (sh, ksh, zsh)	OS9	#!/bin/sh #!/bin/ksh #!/bin/zsh	No
TCL	OS9	#!/usr/bin/tclsh	Yes
Python	SmartScripts	#!/usr/pkg/bin/python	Yes
Perl	SmartScripts	#!/usr/pkg/bin/perl	Yes
Ruby	SmartScripts	#!/usr/pkg/bin/ruby	Yes

S3100 does not support Ruby, and the path to python is #!/f10/flash/pkg/usr/pkg/bin/python

### OS10

Language	Where	Path to binary	Remote debugging?
Unix/Linux shells (sh, bash)	OS10	#!/bin/sh #!/bin/bash	No
Python	OS10	#!/usr/bin/python	Yes
Perl	OS10	#!/usr/bin/perl	Yes

Note that you can install other languages in OS10 using Linux package management tools.

There are several IDE's available with good support for scripting languages, such as Eclipse, Visual Studio and Komodo. Choosing an IDE is a personal preference. In this guide, we will use two: The free version of Microsoft Visual Studio and ActiveState's Komodo IDE.

- **Eclipse:** Open source and free. Supports many languages via a plugin architecture. There are plugins to edit and debug TCL, Perl, Python, Java, etc. Requires some time to master its possibilities: <https://eclipse.org>
- **Visual Studio Community:** Free version of Microsoft's capable Visual Studio. It supports edit and debugging of Python: <https://www.visualstudio.com>

- **Komodo IDE:** Commercial tool. It has very good support for scripting languages. Easy and intuitive: <https://www.activestate.com/komodo-ide>

Install your favorite IDE. The procedure between tools should not differ much.

### Test scripts

Mount the share from Windows and from the switches as described earlier.

Create two new files from the PC, a Python and a Perl script.

These scripts will not run if they use MSDOS line terminations. Configure your editor to use Linux style.

#### test.py

```
#!/usr/pkg/bin/python

if __name__=="__main__":
    count = 5
    i = 1
    print "Start"
    while i <= count:
        print " Python",i
        i = i + 1

    print "End"
```

#### test.pl

```
#!/usr/pkg/bin/perl

$val = 5;
print "Start\n";
for (my $i = 1; $i <= $val; $i++) {
    print " Perl $i\n";
}
print "End\n";
```

In OS9, you must have SmartScripts installed in order to execute Python, Perl and Ruby scripts:

```
Dell# show packages system
```

```
Package Information
```

Unit	Package Name	Version	Status
1	SMARTSCRIPTS	9.11(0.0)	Installed

```
Dell# script exec /f10/mnt/nfs/share/test.py
```

```
Start
```

```
Python 1  
Python 2  
Python 3  
Python 4  
Python 5
```

```
End
```

```
Dell# script exec /f10/mnt/nfs/share/test.pl
```

```
Start
```

```
Perl 1  
Perl 2  
Perl 3  
Perl 4  
Perl 5
```

```
End
```

## Komodo IDE

Komodo IDE is the leading and affordable Integrated Development Environment for Python, Perl, TCL, and other dynamic languages. Komodo IDE is commercialized by ActiveState:

<http://www.activestate.com/komodo-ide>

It has versions for Windows, Mac, and Linux. Komodo IDE has per user licensing: the same license covers any platform and any computer owned by the user.

Refer to the ActiveState documentation for the installation procedure.

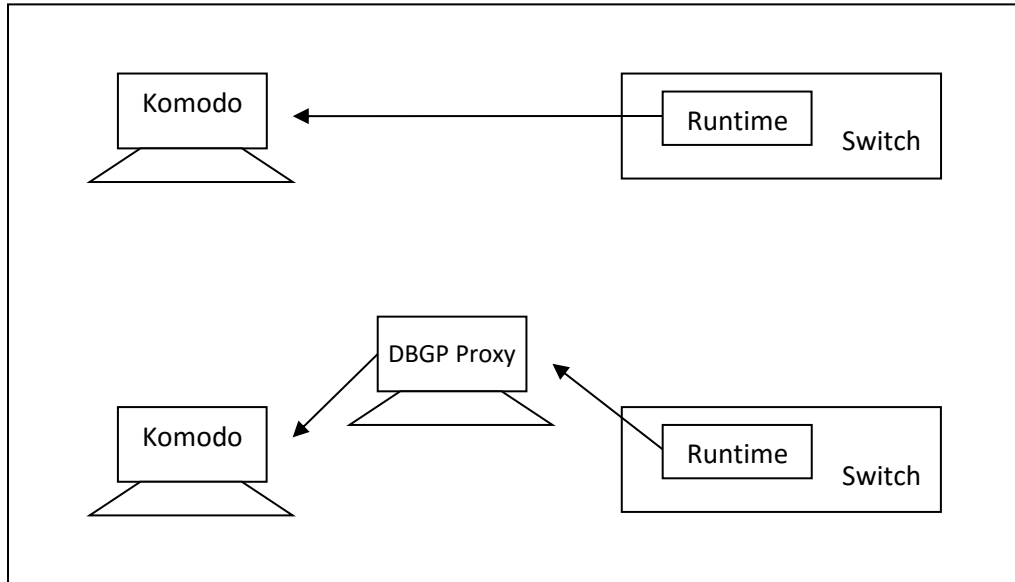
## Debugging with Komodo IDE

Komodo IDE supports local as well as remote editing and debugging of scripts in several dynamic languages, including Perl and Python. Actually, remote debugging is a native feature of these languages, but unlike Java, the Perl and Python runtimes initiate the connection with the remote debugger.

Komodo IDE uses customized versions of the runtime libraries that enable remote debugging. The installation includes these Remote Debugging Client packages. They are also freely available from:

<http://code.activestate.com/komodo/remotedebugging>

After installing Komodo IDE, download the Perl and Python Remote Debugging Client packages for Linux (32bits) to the network share. Note that ActiveState does not support a TCL Remote Debugging Client for NetBSD (OS9).



Sometimes, firewall rules prevent a direct connection with the debugger, or the user needs to run multiple remote debugging sessions. In these cases a *Debugger Proxy* can be used to forward debug traffic between the switch and the debugger.

Komodo includes a DBGP Proxy written in Python at <Komodo>\lib\support\dbgp\bin.

All documentation can be found here:

<http://docs.activestate.com>

<http://docs.activestate.com/komodo/8.5/debugger.html>

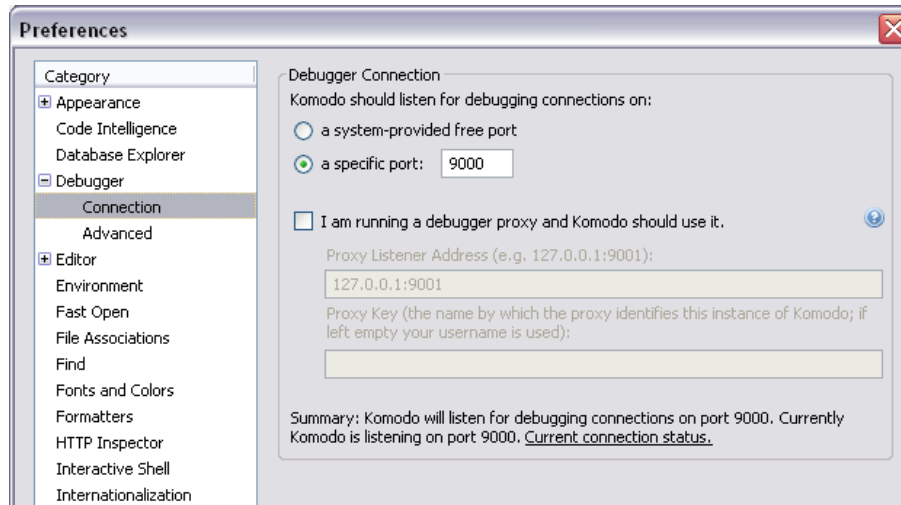
## Debugging Python

### Configuration

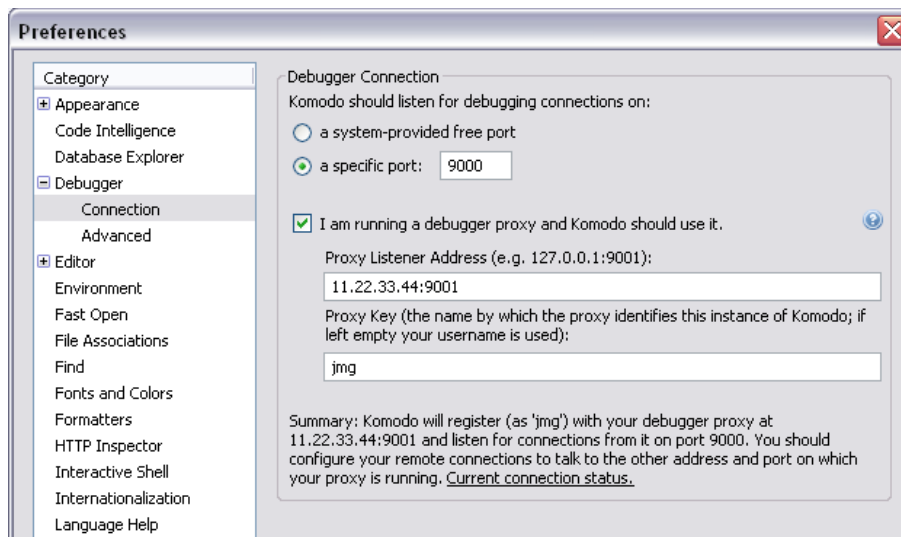
On the machine running Komodo:

1. Open Komodo. Click on the menu **Edit -> Preferences -> Debugger -> Connection**. Select a port, e.g., 9000. Exit the Preferences dialog





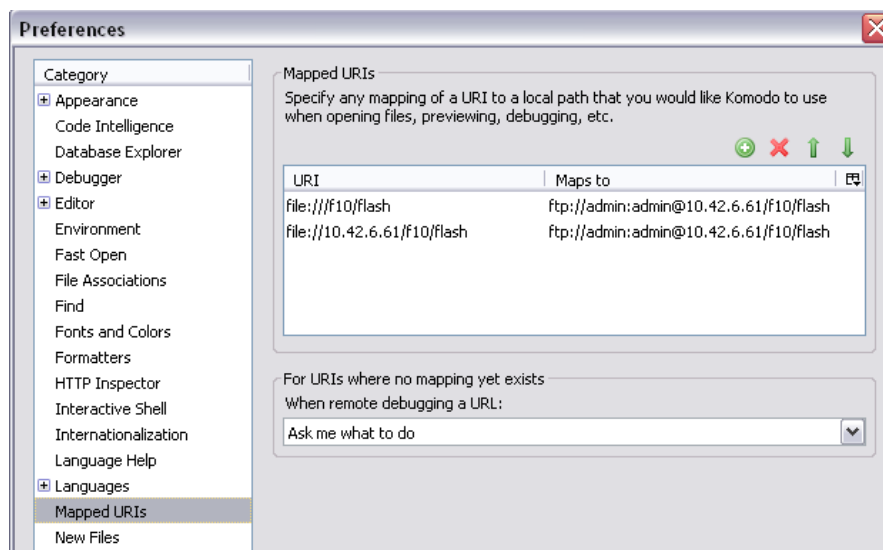
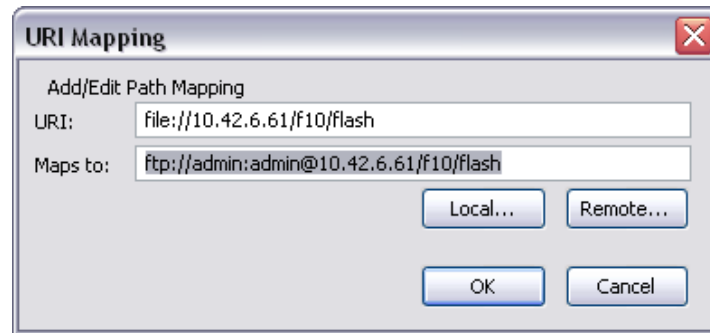
If Komodo IDE connects the switch via a DBGP Proxy activate the box and set the address and port, and any unique string for the Proxy Key:



2. Click on the menu **Debug** and activate the option **Listen for Debugger Connections**.
3. Create a new rule in the PC's firewall to allow incoming connections to TCP-9000
4. Komodo has a very versatile mount-like feature called URI Mapping. It supports ftp, sftp, and scp. When a remote debugging session is established, the runtime sends a string with the format:  
`file://<hostname or ip-address>/<script absolute path>`  
then, you create a map telling Komodo how to retrieve the file with the format:  
`<protocol>://<user>:<password>@<switch-ip-address>/<path>`  
For example, if the script is located at or under /f10/flash, click on the menu **Edit -> Preferences -> Mapped URIs** and enter two maps for  
`file:///<script absolute path>`

and

`file://<hostname or ip-address>/<script absolute path>`



OS9

On the switch:

1. Create a directory on the switch flash

```
Dell# script exec mkdir args Komodo
```

2. Copy the remote debugging package (Tarball format) to the Komodo directory in the switch flash. Untar the file

```

Dell# cd Komodo
Dell# copy nfsmount://share/Komodo-PythonRemoteDebugging-8.5.4-86985-
linux-x86.tar.gz kpython.tar.gz
!
191889 bytes successfully copied
Dell# script exec tar args "zxf kpython.tar.gz"
<Ignore timestamp errors>
Dell# delete kpython.tar.gz no-confirm
Dell# dir
Directory of flash:/Komodo
...
Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
Dell# cd

```

3. Access the switch's shell prompt. Add the path of the remote debugging package to the environment variable PYTHONPATH, and the bin path to PATH. For example:

```

Dell# start shell
<use same CLI credentials>
$ vi .profile
<Append these lines at the end>
KOMODO_PYTHON=/flash/Komodo/Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
PATH=${PATH}:${KOMODO_PYTHON}
if [ -z ${PYTHONPATH} ]; then
    export PYTHONPATH=${KOMODO_PYTHON}/pythonlib
else
    PYTHONPATH=${PYTHONPATH}:${KOMODO_PYTHON}/pythonlib
fi
$ exit
Dell#

```

If a DBGP Proxy is used, the remote address must point to the proxy and the variable DBGP\_IDEKEY must be set to the Proxy Key:

```
export DBGP_IDEKEY="jmg"
```

4. To save some typing while invoking the Python scripts, create this shell script and copy it to the debugger path

**pydbgp.sh**

```

#!/bin/ksh
PYDBGP=${0%`basename $0`}pydbgp
exec python -S $PYDBGP ${1+"$@"}

```

```
Dell# cd Komodo/Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
Dell# copy nfsmount://share/pydbgp.sh pydbgp.sh
!
76 bytes successfully copied
Dell# dir
Directory of flash:/Komodo/Komodo-PythonRemoteDebugging-8.5.4-86985-
linux-x86
...
pydbgp.sh
Dell# cd
```

### OS10

On the switch:

1. Create a directory on the switch flash

```
OS10# system bash
admin@OS10:/config/home/admin$ cd
admin@OS10:~$ mkdir Komodo
```

2. Download the remote debugging package (Tarball format) and untar the file

```
admin@OS10:~$ cd Komodo
admin@OS10:~/Komodo$ wget
http://downloads.activestate.com/Komodo/releases/archive/8.x/8.5.4/re
motedebugging/Komodo-PythonRemoteDebugging-8.5.4-86985-linux-
x86.tar.gz
admin@OS10:~/Komodo$ tar xzf Komodo-PythonRemoteDebugging-8.5.4-
86985-linux-x86.tar.gz
admin@OS10:~/Komodo$ rm Komodo-PythonRemoteDebugging-8.5.4-86985-
linux-x86.tar.gz
admin@OS10:~/Komodo$ cd
```

3. Access the switch's shell prompt. Add the path of the remote debugging package to the environment variable PYTHONPATH, and the bin path to PATH. For example:

```
admin@OS10:~$ vi .bashrc
<Append these lines at the end>
KOMODO_PYTHON=/home/admin/Komodo/Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
PATH=${PATH}:${KOMODO_PYTHON}
if [ -z ${PYTHONPATH} ]; then
    export PYTHONPATH=${KOMODO_PYTHON}/pythonlib
else
    PYTHONPATH=${PYTHONPATH}:${KOMODO_PYTHON}/pythonlib
fi
admin@OS10:~$ exit
OS10#
```

If a DBGP Proxy is used, the remote address must point to the proxy and the variable DBGP\_IDEKEY must be set to the Proxy Key:

```
export DBGP_IDEKEY="jmg"
```

4. To save some typing while invoking the Python scripts, create this shell script and copy it to the debugger path

#### **pydbgp.sh**

```
#!/bin/bash
PYDBGP=$(dirname $0)/pydbgp
exec python -S $PYDBGP ${1+"$@"}
```

```
OS10# system bash
admin@OS10:/config/home/admin$ cp /share/pydbgp.sh Komodo/Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
admin@OS10:/config/home/admin$ pydbgp.sh --help
    pydbgp [dbgp_args_below] script.py [script_args]
    -d hostname:port to debug a script
    ...
admin@OS10:/config/home/admin$ exit
OS10#
```

#### Debug session

From the switch shell invoke the auxiliary shell script with the -d option, and optionally, the -k if the switch has to contact a DBGP Proxy:

```
$ pydbgp.sh -d <ip-address>:<port> [-k key] <script> [arguments]
```

The following command returns all options:

```
$ pydbgp.sh -help
```

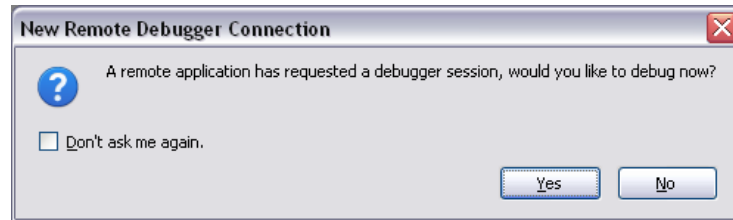
OS9

```
Dell# start shell
<use same CLI credentials>
$ pydbgp.sh -d 10.10.16.10:9000 /f10/mnt/nfs/share/test.py
```

OS10

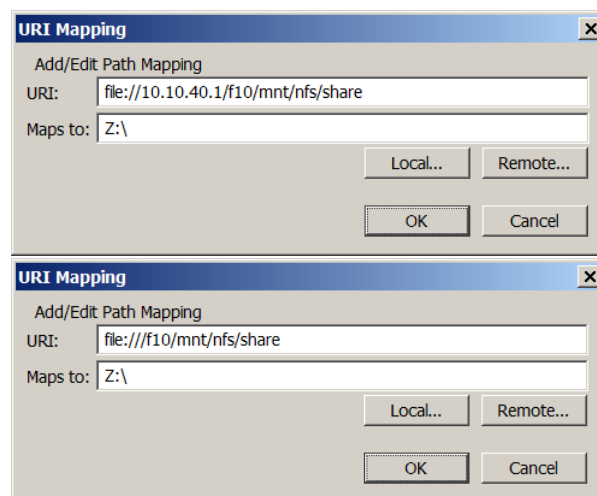
```
OS10# system bash
admin@OS10:~$ pydbgp.sh -d 10.10.16.10:9000 /share/test.py
```

The Python process will try to contact the machine running Komodo. Komodo will display a notification asking the user to accept the debug session:

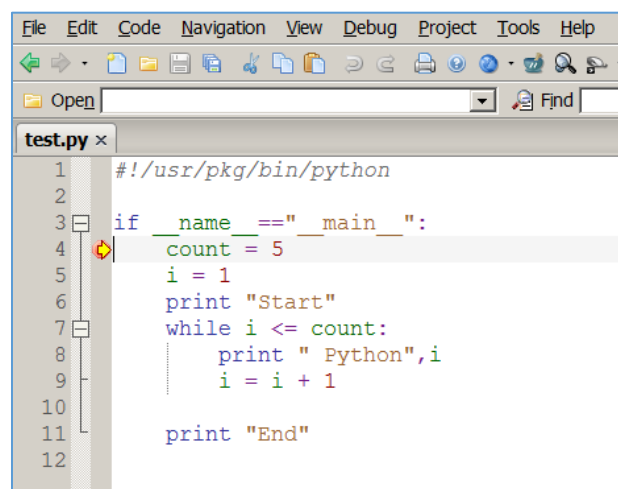


Click Yes. Next, Komodo will complain that it does not have a mapping for the URL:  
`file:///f10/mnt/nfs/share/test.py`

Create two entries both mapped to the Network share's directory where the scripts are:



Now, Komodo will retrieve the script and show the first line to execute. Komodo will not start the script until you enter F5 (GO). You can set breakpoints, stop the execution, read and write variables, etc.



And more importantly, since the script is stored in the Network share, you can edit, save and try it directly, which allows you to test and optimize your code in a true IDE environment.

## Debugging Perl

### Configuration

The steps to configure Komodo IDE for Perl are identical to the Python case. The only difference is on the switch, you need to install the Perl remote debugging package.

#### OS9

On the switch:

1. Copy the remote debugging package (Tarball format) to the Komodo directory in the switch flash. Untar the file

```
Dell# cd Komodo
Dell# copy nfsmount://share/Komodo-PerlRemoteDebugging-8.5.4-86985-
linux-x86.tar.gz kperl.tar.gz
!
109143 bytes successfully copied
Dell# script exec tar args "zxf kperl.tar.gz"
<Ignore timestamp errors>
Dell# delete kperl.tar.gz no-confirm
Dell# dir
Directory of flash:/Komodo
...
Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86
Komodo-PythonRemoteDebugging-8.5.4-86985-linux-x86
Dell# cd
```

2. Access the switch's shell prompt. Set the following environment variables: PERL5LIB and PERLDB\_OPTS in the users's profile file.

PERL5LIB must be set to the path to the remote debugging package

PERLDB\_OPTS must contain the string

"RemotePort=<komodo-ip-address>:<komodo-port> async=1"

async=1 enables "Break Now" on Komodo

For example:

```
Dell# start shell
<use same CLI credentials>
$ vi .profile
<Append these two lines at the end>
export PERL5LIB=/flash/Komodo/Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86
export PERLDB_OPTS="RemotePort=10.10.16.10:9000 async=1"
$ exit
Dell#
```

If a DBGP Proxy is used, the remote address must point to the proxy and the variable DBGP\_IDEKEY must be set to the Proxy Key:

```
export DBGP_IDEKEY="jmg"
```

#### Note

This procedure works on any platform running ActiveState's remote packages.

However, the syntax on Windows does not use double quotes:

```
> set PERL5LIB=C:\Program Files\ActiveState\Komodo IDE 6\lib\support\dbgp\perl5lib
> set PERLDB_OPTS=RemotePort=10.10.16.10:9000 async=1
```

### OS10

On the switch:

1. Create a directory on the switch flash

```
OS10# system bash
admin@OS10:/config/home/admin$ cd
admin@OS10:~$ mkdir Komodo
```

2. Download the remote debugging package (Tarball format) and untar the file

```
OS10# system bash
admin@OS10:/config/home/admin$ cd ~/Komodo
admin@OS10:~/Komodo$ wget
http://downloads.activestate.com/Komodo/releases/archive/8.x/8.5.4/remotedebugging/Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86.tar.gz
admin@OS10:~/Komodo$ tar xzf Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86.tar.gz
admin@OS10:~/Komodo$ rm Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86.tar.gz
admin@OS10:~/Komodo$ cd
```

3. Access the switch's shell prompt. Add the path of the remote debugging package to the environment variable PYTHONPATH, and the bin path to PATH. For example:

```
admin@OS10:~$ vi .bashrc
<Append these lines at the end>
export PERL5LIB=/home/admin/Komodo/Komodo-PerlRemoteDebugging-8.5.4-86985-linux-x86
export PERLDB_OPTS="RemotePort=10.10.16.10:9000 async=1"
admin@OS10:~$ exit
OS10#
```

### Debug session

From the switch shell invoke the Perl script with the -d option:

```
$ perl -d <script> [arguments]
```

### OS9

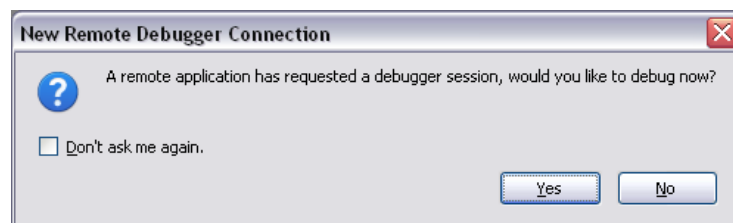


```
Dell# start shell
<use same CLI credentials>
$ perl -d /f10/mnt/nfs/share/test.pl
```

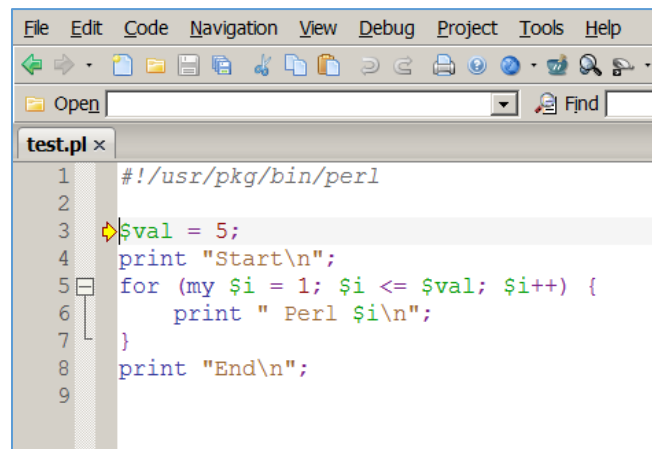
## OS10

```
OS10# system bash
admin@OS10:~$ perl -d /share/test.pl
```

The Perl process will try to contact the machine running Komodo. Komodo will display a notification asking the user to accept the debug session:



Click Yes. If you have created the URL mappings then Komodo will display the script ready to start:



Like Python, Komodo IDE will retrieve the main script and show the program counter in the first instruction. If the script is found under one of the mapped URIs, then you can edit the script from the IDE.

## Visual Studio

Visual Studio is the flagship and richest IDE for implementing Microsoft's technologies (Windows, Office, Azure, Web). For the past several releases, Visual Studio has added support for Linux environments, Mobile apps and Python.

Microsoft offers a free edition called Visual Studio Community, which contains enough functionality to develop, debug and troubleshoot Windows and Linux applications.

<https://www.visualstudio.com>

## Debugging Python

Visual Studio can debug Python applications running remotely, using a connector: “Python Tools for Visual Studio Debugger”, **ptvsd** library. One drawback of this method is that the source code of your Python scripts have to be modified in order to invoke the ptvsd agent.

See more details here:

<https://docs.microsoft.com/en-us/visualstudio/python/debugging-cross-platform-remote>

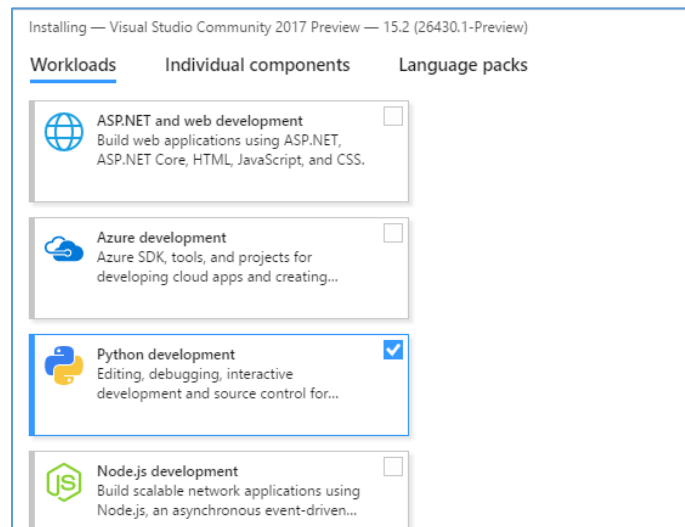
Download the latest version of the ptvsd library from <https://pypi.python.org/pypi/ptvsd>

At the time of this writing, Microsoft announced that Python will be included and installed with Visual Studio 2017. Users of Visual Studio 2015 and earlier versions have to download and install a separate Visual Studio Add-on “Python Tools for Visual Studio” (PTVS) to enable Python. If you are using Visual Studio 2015, install the latest PTVS from <https://github.com/Microsoft/PTVS>

Per this blog, Python will be available on Visual Studio 2017 around June 2017. For now, it is only available on the Visual Studio 2017 Preview versions

<https://blogs.msdn.microsoft.com/pythonengineering/2017/03/07/python-support-in-vs2017>

Visual Studio 2017 Community Preview: <https://www.visualstudio.com/vs/preview>



## Configuration

OS9

On the switch:

1. Create a directory on the switch flash

```
Dell# script exec mkdir args VisualStudio
```

2. Download the latest version of **ptvsd** from <https://pypi.python.org/pypi/ptvsd>. Unzip it into the Network share
3. Copy the ptvsd files to the VisualStudio directory in the switch flash

```
Dell# cd VisualStudio
Dell# script exec cp args "-r /f10/mnt/nfs/share/ptvsd-3.1.0rc1 ."
Dell# dir
Directory of flash:/VisualStudio
...
ptvsd-3.1.0rc1
Dell# cd
```

4. Access the switch's shell prompt. Add the path of the remote debugging package to the environment variable PYTHONPATH. For example:

```
Dell# start shell
<use same CLI credentials>
$ vi .profile
<Append these lines at the end>
PTVSD_PYTHON=/flash/VisualStudio/ptvsd-3.1.0rc1
if [ -z ${PYTHONPATH} ]; then
    export PYTHONPATH=${PTVSD_PYTHON}
else
    PYTHONPATH=${PYTHONPATH}:${PTVSD_PYTHON}
fi
$ sysctl -a | grep net.inet.ip.lowportmax
net.inet.ip.lowportmax = 34999
$ sysctl -a | grep net.inet.ip.anonportmin
net.inet.ip.anonportmin = 49152
$ exit
Dell#
```

While you are still in the shell, run `sysctl` to find out the highest value of the reserved ports and the lowest value of the ephemeral ports (eg., 34999 and 49152, respectively)

## OS10

On the switch:

1. Create a directory on the switch flash

```
OS10# system bash
admin@OS10:/config/home/admin$ cd
admin@OS10:~$ mkdir VisualStudio
```

2. Download the remote debugging package (Tarball format) and untar the file

```
OS10# system bash
admin@OS10:/config/home/admin$ cd ~/VisualStudio
admin@OS10:~/VisualStudio$ cp -r /share/ptvdsd-3.1.0rc1 .
admin@OS10:~/VisualStudio$ cd
```

3. Access the switch's shell prompt. Add the path of the remote debugging package to the environment variable PYTHONPATH, and the bin path to PATH. For example:

```
admin@OS10:~$ vi .bashrc
<Append these lines at the end>
PTVSD_PYTHON=/home/admin/VisualStudio/ptvdsd-3.1.0rc1
if [ -z ${PYTHONPATH} ]; then
    export PYTHONPATH=${PTVSD_PYTHON}
else
    PYTHONPATH=${PYTHONPATH}:${PTVSD_PYTHON}
fi
admin@OS10:~$ exit
OS10#
```

### Script preparation

Add the highlighted code to test.py right before the entry point. These new lines will cause the Python Tools for Visual Studio Debugger (**ptvdsd**) module to wait 10 seconds for Visual Studio to connect. Then, there is a small pause (5 seconds) to allow you set some breakpoints in the code.

After 10 seconds, if Visual Studio has not established a debugging session, the script will run normally.

The `enable_attach` function defines two parameters. The first is a secret ("jmg"). It is a simple string that identifies the connection, similar to the community name in SNMPv2. The second is the address and port to listen. The port must be any value within the range shown above, eg., 39,900

### test.py

```
#!/usr/pkg/bin/python

import time
import ptvsd

ptvsd.enable_attach('jmg', ('', 39900))
ptvsd.wait_for_attach(10)
time.sleep(5)

if __name__=="__main__":
    count = 5
    i = 1
    print "Start"
    while i <= count:
        print " Python",i
        i = i + 1

    print "End"
```

#### Debug session

Open **test.py** from the Network share in Visual Studio.

From the switch shell invoke the python script as usual:

OS9

```
Dell# start shell
<use same CLI credentials>
$ /f10/mnt/nfs/share/test.py
```

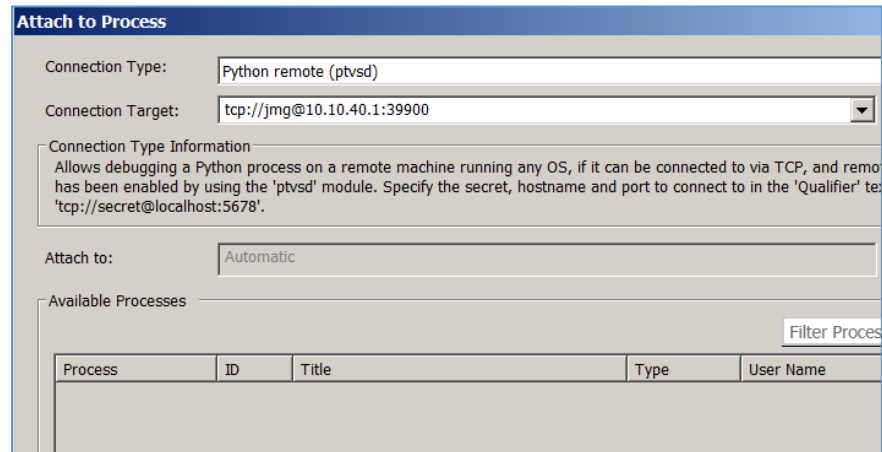
OS10

```
OS10# system bash
admin@OS10:~$ /share/test.py
```

Now, you have 10 seconds to establish a connection from Visual Studio. Click on **Debug -> Attach to Process**. In the Connection Type, select “Python remote (ptvsd)”. In the Connection Target field enter a string with the format:

tcp://secret@hostname:port

For example, tcp://jmg@10.10.40.1:39900



**Attach to Process**

Connection Type: Python remote (ptvsd)

Connection Target: tcp://jmg@10.10.40.1:39900

Connection Type Information  
Allows debugging a Python process on a remote machine running any OS, if it can be connected to via TCP, and remote debugging has been enabled by using the 'ptvsd' module. Specify the secret, hostname and port to connect to in the 'Qualifier' text box: 'tcp://secret@localhost:5678'.

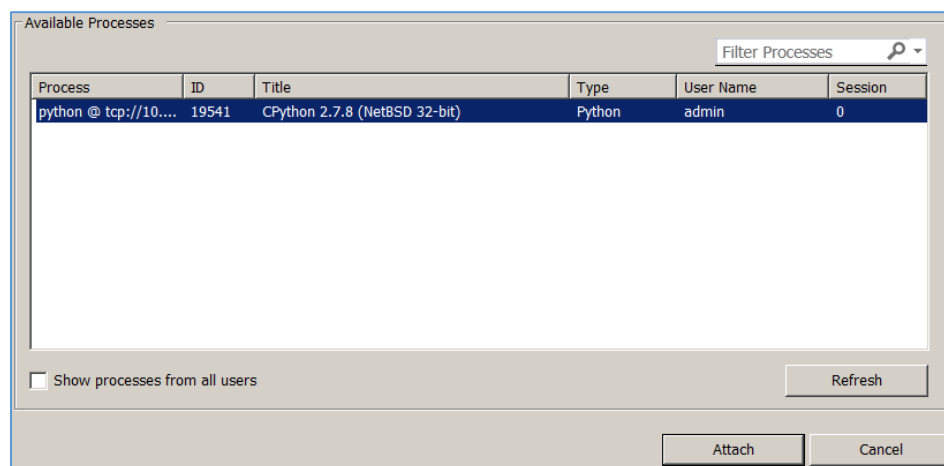
Attach to: Automatic

Available Processes

Process	ID	Title	Type	User Name
---------	----	-------	------	-----------

Filter Processes

While the focus is still on the “Connection Target” edit box, press “Enter”. If the handshake is successful, you will see details of the process running on the switch. Select it and click on “Attach”



Available Processes

Filter Processes

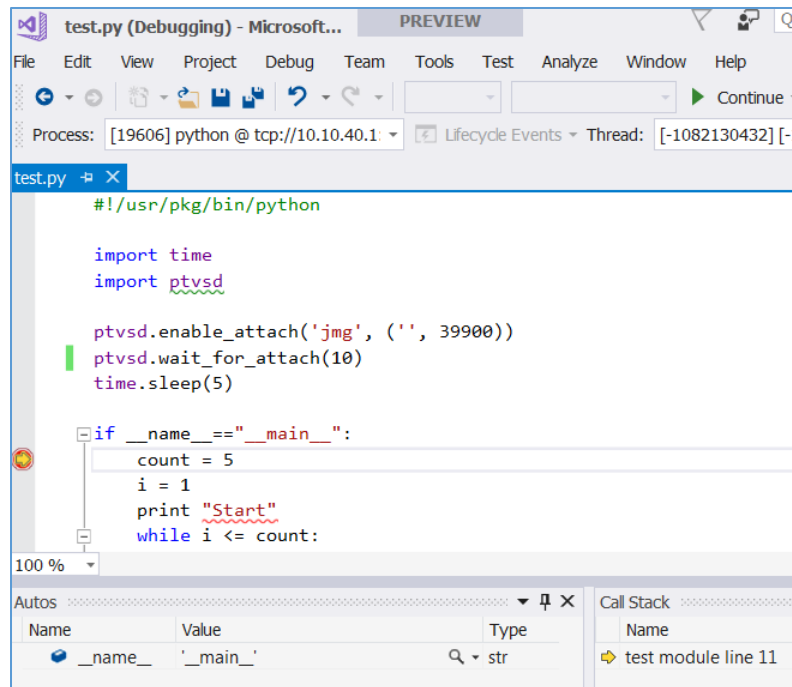
Process	ID	Title	Type	User Name	Session
python @ tcp://10....	19541	CPython 2.7.8 (NetBSD 32-bit)	Python	admin	0

☐ Show processes from all users

Refresh

Attach Cancel

Now you have 5 seconds (the call to `sleep`) to set a breakpoint of pause the execution.



test.py (Debugging) - Microsoft... PREVIEW

File Edit View Project Debug Team Tools Test Analyze Window Help

Process: [19606] python @ tcp://10.10.40.1 Lifecycle Events Thread: [-1082130432] [-

test.py

```
#!/usr/pkg/bin/python

import time
import pty

pty.enable_attach('jmg', ('', 39900))
pty.wait_for_attach(10)
time.sleep(5)

if __name__ == "__main__":
    count = 5
    i = 1
    print "Start"
    while i <= count:
```

100 %

Autos

Name	Value	Type
__name__	'__main__'	str

Call Stack

Name
test module line 11