opConfig User Manual (Version 1.x)

Configuration Options

Credential Sets

Credentials for all connections made by opConfig are stored in conf/credential_sets.nmis. Each connection stores the name of a credential set in it's settings and when a connection is made the username/password for that credential set is looked up and used. This helps limit the locations where credentials are stored and makes it easier to change credentials in bulk.

The layout of the credential set file is very simple.

```
%hash = (
 'empty' => { username => '', password => '' },
 'set1' => { username => 'testuser2', password => 'testpass2' }
);
```

Connections

Connections define which devices/nodes opConfig can connect to. Each connection has 2 common hashes that help layout how the connection will be treated. The definition of connection_info is fairly rigid, any other extra hash can be defined and information in that hash can be used to filter commands to run on that device in the command sets. More will be explained on this later.

The connection info hash inside each device/node tells opConfig how to connect to the device and what "language/personality" it has. An example connection_info hash:

```
'connection_info' => {
 'transport' => 'Telnet',
 'credential_set' => 'set3',
 'personality' => 'ios',
 'node' => 'asgard',
 'host' => '192.168.88.254',
 'priveleged_credential_set' => 'set3',
 'force_active' => 'true'
 }
```

An explanation of connection_info:

```
node => the name of the node in NMIS
host => the ip of the device/node
transport => SSH or Telnet
personality => catos, extremeos, foundry, hp, ios, junos, nortel, pixos, bash or csh
credential_set => any of the credential keys defined in credential_sets.nmis
force_active => by default only nodes in NMIS that are "active" and "collect" are run, setting this to true
forces the device/node to be run
```

The os_info hash describes a bit about the operating system used by the device/node. An example os_info hash:

```
'os_info' => {
 'featureset' => 'Unknown',
 'version' => '12.4(25f)',
 'platform' => '1841',
 'train' => '12.4',
 'major' => '12.4',
 'os' => 'IOS',
 'image' => 'C1841-ADVENTERPRISEK9-M'
 }
```

Explanation of os_info:

```
os => name of os
version => version of os
```

The info in os_info is defined by the discovery function of opConfig. The settings defined in it are used to filter command sets as will be seen in the command sets section.

When opConfig does it's own discovery, it maps the operating system into a personality, here is that map:

```
my $os_personality_map = {
 'AlterPath' => 'unknown',
 'CatOS' => 'catos',
 'Cat2820' => 'catos',
 'Cat1900' => 'catos',
 'Darwin' => 'bash',
 'ExtremeXOS' => 'extremexos',
 'FastHub' => 'unknown',
 'Foundry' => 'foundry',
 'HP' => 'hp',
 'IOS' => 'ios',
 'junos' => 'unknown',
 'NAMOS' => 'unknown',
 'NetScout' => 'unknown',
 'Nokia' => 'unknown',
 'nortel' => 'unknown',
 'ONS' => 'unknown',
 'PIX' => 'pixos',
 'fwsm' =>'fwsm',
 'fwsm3' => 'fwsm3',
 'pixos7' => 'pixos7',
 'Solaris' => 'bash',
 'SwitchProbe' => 'unknown',
 'Linux' => 'bash',
 'VPN' => 'unknown',
 'Windows' => 'unknown',
 'Z_Unknown_Unix' => 'csh'
};
```

Command Sets

Each command set defines information used to filter out which devices/nodes the commands will be run on. It also defines the actual commands to be run and if the commands should be grouped together into a single session/connection or be run on their own.

Here is an example command set, I will break it down in the sections that follow:

```
'IOS_DAILY' => {
 'os_info' => {
   'version' => '/12.2|12.4|15.0/',
   'os' => 'IOS'
 },
 'aging_info' => {
   'age' => 'forever'
 },
 'scheduling_info' => {
   'run_commands_on_separate_connection' => 'false'
 },
 commands => [
 {
   'tags' => 'config,version,troubleshooting, detect-change',
   'command' => 'show version',
   'privileged' => 'false',
   'multipage' => 'true',
   'run_command_on_separate_connection' => 'false',
   'command_filters' => [
     '/uptime is/'
   ]
 }
}}
```

Defining which connections should be used by a command set

Filtering by OS info

```
#filter using a regular expression:
'os_info' => {
 'version' => '/12.2|12.4|15.0/',
 'os' => 'IOS'
},
# or just the specific os and that's all:
'os_info' => {
 'os' => 'Linux'
},
```

A regular expression can be used in any filtering field. Any hash or field can be defined in a connection and then filtered on in the command set (not just the ones that have been shown).

Making opConfig detect changes in a command output

Not all output is worthy of change detection, configuration information generally is while performance information generally is not. The configuration for a router, for example, is but the usage on an interface is not. Commands that have the tag **detect-change** will have change detection run on them by opConfig. If the output is different a new revision is stored along with the changes that were made. If it is not defined a new revision is stored (if the output is different) but no changes are tracked (and will not be reported in the GUI as a change).

```
'tags' => 'detect-change',
```

There is one more issue to resolve. Not all changes found may be considered important or relavent. To ignore unimportant changes a set of command filters can be added to a command:

```
'command_filters' => [
 '/uptime is/'
]
```

If the regular expression matches a change that change is considered unimportant and will not be tracked by the change management system (and so the GUI will not report it as a change)

Grouping commands into sessions

By default all commands for a device will be run one after another on the same session/connection to that device. This will generally save time as connecting to the device often takes longer than running all the commands on it. This is not always the case and there will be certain commands that run for a longer period of time that should have their own connection.

If you would like to change the default behaviour and have opConfig run each command on it's own session/connection, opCommon.nmis can be modified:

```
# to run each command in it's own session, change this to true
 'opconfig_run_commands_on_separate_connection' => 'false',
```

It makes a lot more sense to tell opConfig to run a set of commands on their own or, in most cases, just a single command on it's own. Running a command or set of commands on their own session/connection:

```
#to run all commands in a command set define this key to be true
'scheduling_info' => {
 'run_commands_on_separate_connection' => 'true'
},

# to run just a specific command on it's own
commands => [
 {
 'command' => 'show version',
 'run_command_on_separate_connection' => 'true',
 }]
```

Aging

Aging is not currently in use, look for more info about this in later versions

Usage

opConfig-cli

Information on what commands are supported is printed when no options are specified. All options support debug=true for debug output and debug=9 for extremely verbose output.

Listed below are the possible options for act=

discover, Automatically discover new connections

Grabs all nodes from configured NMIS server and attempts to add all nodes that are enabled and have collect=true.

Options

- disable_test=true – does not attempt to connect and adds the all nodes with the credential_set for each connection set to an invalid value
- force_active=true – ignores settings in NMIS and adds all nodes not just the ones that are active and being collected
- nodes=node1,node2,etc – only attempts to discover nodes listed

test_connect, Testing a connection

opConfig-cli can be used to test connections to help debug situations that don't make any sense.

An example of how it can be used:

```
bin/opConfig-cli.pl act=test_connect host=192.168.88.254 transport=Telnet personality=ios username=testuser
password=testpass
```

The options for transport and personality are given above.

It is also possible to test an existing connection from the connections.nmis file by specifying node=node_name , if any options are specified on the command line along with the node they will override the settings loaded from the connections.nmis file.

command="some command" can also be specified to test the output of a specific command.

run_command_sets, Running commands on devices

This command will run all command sets for all nodes (by default).

Options:

- nodes=node1,node2,etc -- only the specified nodes will be run against all command sets.
- command_set_names=command_set1,command_set2,etc – only run the specified command sets, this will still only run nodes that match the command sets criteria
- tags=tag1,tag2,etc – the matching nodes are filtered by the tags specified, if any tag matches the node will be run

get_command_output, Get the last output from a command for a node

Requries node=node_name command="command name" and returns output collected from the last run of this command

diff_command_outputs, Diff two revisions

Shows the diff from the output of 2 revisions of stored output (does not run them, only queries). The command line would look similar to get_command_output with the edition of revision_1= and revision_2=

eg. act=diff_command_outputs node=node1 command="show run" revision_1=10 revision_2=13

create_indexes, Create indexes in DB

This command is explained in the installation docs. If you end up losing your database you will want to re-run this command to ensure good performance. It is safe to run this command more than once.