

**NetConnect**  
**Network Authentication Tool**  
by Chris Mason <chris@netnix.org>

url <http://netnix.org/netconnect>

**User Guide**  
**v1.48**

## Contents

1.	Introduction .....	3
2.	Minimum Requirements .....	3
3.	Usage.....	3
3.1.	Actions.....	4
3.2.	Options.....	7
3.3.	Title Bar .....	9
3.4.	Shell Return Codes .....	9
3.5.	Examples .....	9
4.	Configuration .....	11
4.1.	Escaping Characters .....	12
4.2.	Connection Methods.....	12
4.3.	Attributes .....	13
4.3.1.	Attribute Encoding .....	13
4.3.1.1.	Changing Passwords .....	14
4.3.2.	Ask Attribute .....	14
4.3.2.1.	One-Time Passwords .....	14
4.3.3.	Execute Attribute .....	14
4.3.4.	Inheritance .....	15
4.3.5.	Node Types.....	16
4.4.	Configuration Schemas .....	16
4.4.1.	Profile Schema .....	16
4.4.2.	Proxy Schema .....	21
4.4.3.	Options Schema .....	23
4.5.	Split Configuration .....	23
4.5.1.	Main Configuration .....	24
4.5.2.	Included Configuration .....	24
4.6.	Configuration Example.....	24
5.	Interactive Mode.....	25
5.1.	Automatic Logoff.....	25
5.2.	Running Scripts.....	26
5.3.	Running Shell Commands .....	26
5.4.	Aliases .....	26
5.5.	Prompt Semantics .....	28
6.	Scripts.....	29
6.1.	Script Object (sO) .....	29
6.2.	Examples .....	33
6.2.1.	Save Script .....	33
6.2.2.	Inventory Script.....	34
6.2.3.	Monitor Script .....	35

## 1. Introduction

NetConnect is a command line tool for automating the login process to routers, switches or UNIX hosts and any intermediate devices or proxies in the path using credentials supplied within a configuration file. It also allows users to run Perl based scripts on a single device or devices in parallel to automate tasks.

## 2. Minimum Requirements

NetConnect doesn't really have any minimum requirements - except a UNIX environment. NetConnect is written in Perl so it requires a Perl runtime environment to be able to run, but 99% of UNIX systems come with Perl as standard or can have Perl installed quite easily. On top of the default Perl installation, NetConnect requires the following additional Perl module to be installed:

- Expect v1.21

Although I specifically mention UNIX, NetConnect is not limited to UNIX alone and was mostly developed under Linux. There is also support under Microsoft Windows, not natively, but under Cygwin which allows you to have a UNIX environment within Windows, which provides a complete run-time environment for NetConnect. Further information on Cygwin can be found at <http://www.cygwin.com>.

## 3. Usage

NetConnect automates the connection and authentication to devices using credentials provided within a configuration file. The configuration file consists of profiles which define a 'username', 'password' as well as a list of devices which use those credentials.

We are going to start off by ignoring the configuration file to demonstrate what NetConnect is able to do. We are going to assume we have already written a configuration file for a small network consisting of a handful of routers.

The NetConnect binary is called 'netc' and if we run NetConnect without any arguments we are going to see the following usage screen:

```
$ netc
```

```
NetConnect v1.48
Network Authentication Tool
by Chris Mason <chris@netnix.org>

Usage: netc <action> [options]
Actions:
```

```

<regex>          Connect to a node within the cfg file
  or file:<node file> Specify multiple nodes from a file (script use only)
  or group:<regex>   Specify multiple nodes by group (script use only)
  or -t <host>      Telnet to an unknown host
  or -s <host>      SSH to an unknown host
  or -rn [regex]    Output nodes selected by regex
  or -rn profile:<regex> Output nodes selected by profile
  or -rn group:<regex> Output nodes selected by group
  or -chpass        Change an encoded password within the cfg file
Options:
-p <regex>        Use specific profile (unknown host use only)
-x <script file>  Execute script on node(s)
-o <option>=<value> Specify script options
-l [logfile]      Write output to logfile
-v                Verbose output

```

NetConnect is able to connect to devices in two different ways; 1) by referencing a node within the configuration file by using a regular expression or 2) by using either the '-t' option to telnet or the '-s' option to SSH – this then uses the credentials from an existing profile within the configuration file.

### 3.1. Actions

Actions are mutually exclusive which means only one may be specified on the NetConnect command line at any one time. They tell NetConnect what action to perform and are explained in detail below:

**<regex>** If a regular expression is specified on the command line then NetConnect will attempt to find a device within the configuration file that matches the case-insensitive expression. The login credentials are provided within the profile which is explicitly stated for that node within the configuration.

NetConnect is only able to interactively connect to a single device, so the regular expression must be explicit enough to only select a single device. As a regular expression can match anywhere within a string you might need to anchor your expressions with a ^ or \$ to anchor to the start or end of the string.

**file:<node file>** Although NetConnect is only able to interactively connect to a single device at any one time, it can connect to devices in parallel when executing scripts. Due to the powerful nature of running scripts some care should be taken when using this option. This option allows you to reference a file which contains a list of devices that you wish the script to be executed on. To ensure you are careful in your device selection this file should contain a list of explicit device names (regular expressions aren't supported).

If you don't want NetConnect to process all nodes in this file then you can comment a node out using a hash character at the beginning of the line.

**group:<regex>** This option follows on from the above option and allows you to specify multiple nodes by matching groups. This allows scripts to be run on multiple nodes at the

same time. Depending on the regular expression depends on how many groups are matched and how many nodes are matched. You can use `^$` to match nodes which aren't associated with groups as well as `.*` to match all nodes.

**-t <host>**

If the user wishes to telnet to a device which isn't in the configuration file then they can use the `-t` command line option to specify a host. NetConnect will then attempt to telnet to that host using either the credentials within the default<sup>1</sup> profile or the credentials within a separate profile specified using the `-p` command line option.

The 'host' can be made up of a hostname or an IP address with an optional port specified using the syntax: 'host:port'.

<sup>1</sup> The default profile is the first profile within the configuration file.

**-s <host>**

If the user wishes to SSH to a device which isn't in the configuration file then they can use the `-s` command line option to specify a host. NetConnect will then attempt to SSH to that host using either the credentials within the default<sup>1</sup> profile or the credentials within a separate profile specified using the `-p` command line option.

The 'host' can be made up of a hostname or an IP address with an optional port specified using the syntax: 'host:port'.

<sup>1</sup> The default profile is the first profile within the configuration file.

**-rn [regex]**

This command line argument tells NetConnect to output the node report which lists all the devices within the configuration file. You may specify an optional regular expression to only output a report containing certain nodes. With the below exception further down (group and profile syntax), the regular expression will be matched against the node name to select what nodes to include.

```
NetConnect v1.48
Network Authentication Tool
by Chris Mason <chris@netnix.org>

[Profile: RADIUS, Type: Cisco, Login: bloggsj]
  [Group: CUSTOMER], [Nodes: 2]
    CE-XX-01 ..... 192.0.2.11
    CE-XX-02 ..... 192.0.2.12

-----
[Profile: RSA-SECURID, Type: Cisco, Login: bloggsj]
  [Group: IPSEC-HUB], [Nodes: 2]
    IPSEC-XX-01 ..... 192.0.2.21
    IPSEC-XX-02 ..... 192.0.2.22

Total Nodes: 4
```

In our example configuration file we have created four devices – two CE routers and two IPSEC hub routers. They are specified under two different login profiles as they use different credentials – the IPSEC hub routers use RSA-SECURID as opposed to the CE routers which use the RADIUS profile.

To ensure the most optimal output, NetConnect will determine how many columns your terminal window is capable of displaying and will display the nodes in a maximum of three columns. Nodes will be displayed alphabetically and vertically. To reduce the amount of columns being displayed you will need to reduce the size of your terminal window. It will also omit profiles from the configuration file that don't contain any nodes unless you also specify the '-v' parameter.

### Matching Groups

If the regular expression starts with 'group:' (or 'grou:', 'gro:', etc) then the regular expression will be used to match nodes based on a group – the regular expression will be matched against the group name (to include spaces the expression will need to be quoted). The following output demonstrates this syntax:

```
$ netc -rn group:cust
```

```
NetConnect v1.48
Network Authentication Tool
by Chris Mason <chris@netnix.org>

[Profile: RADIUS, Type: Cisco, Login: bloggsj]
 [Group: CUSTOMER], [Nodes: 2]
   CE-XX-01 ..... 192.0.2.11
   CE-XX-02 ..... 192.0.2.12

Total Nodes: 2
```

### Matching Profiles

If the regular expression starts with 'profile:' (or 'profil:', 'profi:', etc) then the regular expression will be used to match nodes based on their profile – the regular expression will be matched against the profile name (to include spaces the expression will need to be quoted). The following output demonstrates this syntax:

```
$ netc -rn profile:rsa
```

```

NetConnect v1.48
Network Authentication Tool
by Chris Mason <chris@netnix.org>

[Profile: RSA-SECURID, Type: Cisco, Login: bloggsj]
 [Group: IPSEC-HUB], [Nodes: 2]
   IPSEC-XX-01 ..... 192.0.2.21
   IPSEC-XX-02 ..... 192.0.2.22

Total Nodes: 2

```

### Multiple Matches

The user may specify the '-rn' option multiple times to perform a logical AND as demonstrated in the example below which selects all nodes within the 'PE' group that contain '01' within their name:

```
$ netc -rn group:cust -rn 01
```

```

NetConnect v1.48
Network Authentication Tool
by Chris Mason <chris@netnix.org>

[Profile: RADIUS, Type: Cisco, Login: bloggsj]
 [Group: CUSTOMER], [Nodes: 1]
   CE-XX-01 ..... 192.0.2.11

Total Nodes: 1

```

### -chpass

This option starts the change password function which allows NetConnect to change encoded values within the configuration file. It will prompt you for your old password and will find all occurrences of that within the configuration file. Once it has found all the occurrences it will ask you to confirm which ones you want to be changed and then ask you for a new password.

This option takes away the hassle of having to generate new Base64 encoded strings and updating the configuration file when passwords need to be changed.

## 3.2. Options

Options, as the name suggests are optional and change the default behaviour of NetConnect. Different options may be used in different contexts and not all options can be used in all contexts.

### -p <regex>

This option allows us to specify a profile when connecting to a node when using the '-t' or '-s' command line options. It is not permitted to be used

when a device is selected from the configuration file as the profile for that device is explicit within the configuration.

The argument takes a case-insensitive regular expression to attempt to match a profile within the configuration. Only a single profile can be matched as multiple profiles would become ambiguous.

If this option is omitted then the default profile will be used which would be the first profile within the configuration.

**-x <script file>**

This option allows you to run a script on a node. This script is written in Perl and uses the NetConnect ScriptObject (sO) to interact with the node. Scripts don't require a thorough knowledge of Perl, however to write extensive scripts, a basic knowledge of Perl is required. Scripts are covered in their own section towards the end of this document.

To just check connectivity to a node or nodes then you can use the following trick – this will connect to all nodes that match the group 'cust' and disconnect immediately:

```
$ netc -rn group:cust -x /dev/null
```

By default, the output from commands that are executed within a script won't be visible. To ensure you see what the script is doing make sure you specify the '-v' command line option to enable verbose output.

**-o <option>=<value>**

This option allows you to pass through parameters to scripts. This option may be specified multiple times to pass through multiple options. Quotes will need to be used when spaces are present within the value.

**-l [logfile]**

This option allows you to save the session output to a logfile of your choice. This will output the information which would normally be output to the screen if the '-v' command line argument was used. The actual logfile is optional and if omitted, NetConnect will create the logfile in the current directory using the following format: 'Node\_ID.YYYYMMDD.HHMMSS.log' where 'YYYYMMDD' denotes date and 'HHMMSS' denotes time.

To change the default location of logfiles you may use the 'log\_dir' option in the configuration file which will create logfiles within a specific directory if you don't specify an absolute path.

**-v**

By default, NetConnect won't show the process of connecting and authenticating to devices in the terminal window. If you wish to see what NetConnect is doing then you may specify the '-v' command line option to

activate verbose output. This option can be specified twice to enable Expect debug output for troubleshooting.

Alternatively you can also specify a negative value to suppress the printing of the NetConnect title banner:

```
$ netc -v -1 ...
```

### 3.3. Title Bar

When NetConnect connects to a device it will attempt to change the title bar of the terminal window to show the hostname of the device. Different terminal clients will have different results with this behaviour and this works best if you use PuTTY. Reflection doesn't appear to support this behaviour as the title bar isn't changed and on some versions will cause the session to hang.

NetConnect will attempt to do this if your '\$TERM' environment variable contains the string 'xterm' which is the default for PuTTY. This doesn't happen with Reflection as it uses a '\$TERM' setting of 'vt220' as default.

Also, when running a script on multiple nodes, NetConnect will change the title to show a percentage indicator to inform you how far it has progressed through the nodes.

### 3.4. Shell Return Codes

When NetConnect exits it will set the shell return code to one of the following values depending on why it exited. This will help if running NetConnect from a script (this can be retrieved by looking at '\$?' in the shell).

- 0** No failure – normal exit
- 1** Generic failure – this covers most scenarios and would include for example, invalid command line options.
- 2** Recoverable timeout – this means that NetConnect exited from a node timeout that would have been caught if 'retry\_on\_timeout' was set.
- 3** Fatal timeout – this means that NetConnect exited from a node at a crucial part which implies the node is not responding, etc.
- 4** Script syntax error – this means that there is a syntax error within the script provided.
- 5** Invalid credentials – this means that an invalid username/password has been specified within the configuration file.

### 3.5. Examples

The following section will be based on the information provided within the following snippet taken from a node output report:

```
[Profile: RADIUS, Type: Cisco, Login: bloggsj]
 [Group: CUSTOMER], [Nodes: 2]
   CE-XX-01 ..... 192.0.2.11
   CE-XX-02 ..... 192.0.2.12
-----
[Profile: RSA-SECURID, Type: Cisco, Login: bloggsj]
 [Group: IPSEC-HUB], [Nodes: 2]
   IPSEC-XX-01 ..... 192.0.2.21
   IPSEC-XX-02 ..... 192.0.2.22
```

To help explain the command line options a few examples have been included below to demonstrate some of the more common uses of NetConnect:

```
$ netc -rn
```

The above command will get NetConnect to display the node output report. This contains a summary report of all the devices and profiles defined within the configuration file.

```
$ netc -rn ^CE
```

The above command will get NetConnect to display the node output report, but only showing devices that match the case-insensitive regular expression `^CE`. This regular expression will match all devices that begin with `CE`.

```
$ netc -rn 'CE|IPSEC'
```

The above command will get NetConnect to display the node output report, but only showing devices that match the case-insensitive regular expression `CE|IPSEC`. This regular expression will match all devices that contain either `CE` or `IPSEC`.

```
$ netc -rn 'CE' -rn '01'
```

The above command will get NetConnect to display the node output report, but only showing devices that match the case-insensitive regular expression `CE` that also contain `01` within the name. This regular expression will match all devices that contain `CE` and `01`.

```
$ netc -rn '^(?!.*CE)'
```

The above command uses an advanced regular expression which is called a negative lookahead assertion. This will get NetConnect to display the node output report, but only showing devices which don't contain the expression 'CE'. For information on standard and more advanced regular expressions the following URL can be used as a good reference: <http://perldoc.perl.org/perlre.html>

```
$ netc ce-xx-01
```

The regex of 'ce-xx-01' will match the device 'CE-XX-01' and will connect and authenticate to the device using the credentials supplied within the 'RADIUS' profile as it is defined under this profile.

```
$ netc -t 192.0.2.37
```

The above example will telnet to host '192.0.2.37' using the credentials supplied within the default profile which would be the 'RADIUS' profile as it is the first profile.

```
$ netc -s 192.0.2.53 -p rsa
```

The above example will SSH to host '192.0.2.53' and the profile regular expression will match the 'RSA-SECURID' profile. This will use the login credentials from the 'RSA-SECURID' profile when connecting to the node.

```
$ netc ce-xx-02 -x bgp_ping.ncs -o mtu=1500 -v -l
```

The above example will connect to 'CE-XX-02' based on the regular expression. Once connected it will execute the script 'bgp\_ping.ncs' on the node. The parameter 'mtu' will be passed through to the script with a value of 1500. The '-v' option will turn on verbose output and the '-l' option will save the output into a logfile which is dynamically created. Once the script has finished, NetConnect will disconnect from the node.

#### 4. Configuration

The default location of the configuration file, which is called 'netc.conf' is within your home directory as defined by the '\$HOME' environment variable. If your UNIX username was 'bloggsj' then NetConnect would use the following path for the configuration file: '/home/bloggsj/netc.conf'. This can be overridden using the '\$NETC\_CFG\_FILE' environment variable and if exists will be used as an alternative configuration file instead.

The format of the configuration file is relatively straightforward; it uses squiggly braces to denote sections and commas to delimit attributes with attributes being defined using 'key: value' pairs. All

values should be enclosed in single or double quotes for completeness, but this isn't enforced. The syntax file 'netc.vim' has been supplied in the NetConnect distribution for Vim that can be installed and associated with 'netc.conf' to provide syntax highlighting.

#### 4.1. Escaping Characters

Within the configuration file, some characters need to be escaped to ensure their meaning is passed through and to stop NetConnect from terminating strings prematurely. Regardless of whether you use single or double quotes, the values won't be interpolated; this means you only need to escape things once. This makes writing regular expression easier, but it might make reading them slightly confusing if you are used to double escaping regular expressions when using double quotes. It is recommended to use single quotes whenever possible.

The three main characters which you may need to escape if used within values are: single quotes, double quotes and commas. You will always need to escape commas as they are used to delimit values, but quotes will only need to be escaped if you are using that specific type to enclose your value.

If we ignore the grammatical issues within the configuration example below, we can see that we have had to escape single quotes as we use them to enclose the string, but we haven't had to escape double quotes. The comma in the statement has also had to be escaped:

```
profile: 'Example' {
  password: 'why can\'t I use a "comma", asked the dog?'
}
```

In regular expressions you will only need to escape special characters that you would normally escape if you want their value to be taken literally.

#### 4.2. Connection Methods

NetConnect supports connecting to nodes via telnet or SSH and this is defined within the configuration file as per the example below:

```
profile: 'Example' {
  telnet[: '<group>'] {
    ...
  }
  ssh[: '<group>'] {
    ...
  }
}
```

This will define how NetConnect connects to devices. If NetConnect sees 'telnet' then it will execute the 'telnet' application and if it sees 'ssh' then it will execute the 'ssh' application. You may define an optional group before the brace that the devices will be placed into.

NetConnect needs to know if you are connecting via telnet or SSH as the login sequence is slightly different, but what if 'ssh' wasn't called 'ssh' on your system?

This would present us with a problem as NetConnect wouldn't be able to spawn a session and it would fail. To be platform independent, NetConnect relies on the path to resolve 'ssh' or 'telnet' so we need to ensure that they are located within the path. If 'telnet' was called 'tnt' then you would need to symlink a dummy 'telnet' to 'tnt' to ensure that NetConnect can resolve 'telnet'. The same would be required for 'ssh' if that was also different.

This isn't an ideal solution and is more of a kludge, but for the meantime until I think of a better way to implement this within the configuration schema it can be used to get around this limitation.

### 4.3. Attributes

#### 4.3.1. Attribute Encoding

Within the configuration file, attributes have an optional 'encoding' attribute which allows you to specify that the value has been encoded. This should be used on attributes that could potentially contain sensitive information that you don't want someone looking over your shoulder and being able to see when you are editing the configuration file.

The encoding attribute is not encryption and it isn't meant to be – we are unable to store passwords using one-way hash functions as we need to decode them and use them to login to devices.

NetConnect at the moment supports Base64 encoding which allows the sensitive information to be slightly mutated which makes it difficult for a wandering eye to remember the sequence easily. There are various websites which offer online Base64 encoding and decoding scripts or alternatively you could use the following Perl one-liner to encode a string to Base64:

```
$ perl -MMIME::Base64 -e 'print "Password: "; chomp ($r=<STDIN>); print encode_base64($r)'
```

The 'encoding' attribute is optional and you may put sensitive information into the configuration file without using it, but if you wish to encode the values then you need to use the following format:

```
profile: 'Example' {  
  password: 'Y2hyaXNAbmV0bm14Lm9yZW==', encoding: 'base64'  
}
```

The 'encoding' attribute tells NetConnect that the value specified for the 'password' attribute has been encoded using 'base64'.

#### 4.3.1.1. Changing Passwords

NetConnect supports changing encoded values which are stored within the configuration file using the '-chpass' command line option. If you specify this option then NetConnect will ask you for your old password and will then change all occurrences of that password for a new one that you have provided. This takes away the hassle of having to re-encode new strings as it does all the work for you.

#### 4.3.2. Ask Attribute

The 'ask' attribute can be used if you don't wish to store sensitive passwords within the configuration file or if it is a one-time password using something like an RSA SecurID token.

```
profile: 'Example' {  
  password: '', ask: 'Enter Password: ', type: 'password'  
  or  
  password: '', ask: 'Enter PASSCODE: ', type: 'one-time'  
}
```

You will need to include a blank value for the 'password' attribute or an arbitrary value as it will be overridden with the value the user provides. The 'ask' attribute defines the prompt to use when asking for a password. The optional 'type' keyword can be specified with a value of 'password' to disable echo which stops the password from showing as you type it or 'one-time'. Finally, the optional 'key' attribute allows you to define a sort key to specify the order in which you want to be asked.

If you use the same 'ask' prompt multiple times then NetConnect will only ask the user once and will use the same password multiple times. By default, NetConnect will prompt you in alphabetical order based on the prompt. If you wish to change the order you can place a value in the 'key' attribute to influence the order. If the 'key' attribute is missing then it inherits the value from the 'ask' attribute.

#### 4.3.2.1. One-Time Passwords

RSA SecurID provides one time passwords which change every 60 seconds. This prevents replay attacks where the attacker has managed to obtain the user's password. It is two-factor as it also requires a secret piece of information (the pin) which only the user knows to stop people from stealing your token. In these scenarios, prompting for a password once won't work as it will only work for a single node. By specifying a value of 'one-time' for the 'type' keyword tells NetConnect to prompt the user every time before connecting to a node for a new token. By using a type of 'one-time' will also disable the ability to process nodes in parallel.

#### 4.3.3. Execute Attribute

The 'execute' attribute can be used if you wish to run a shell command and use the output for a value. This is most useful with the 'username' attribute as we can fetch the login username that the user is currently logged in with.

As with the previous example we need to define an initial 'username' attribute although the value is not used as it will be replaced with the output from the shell command.

```
profile: 'Example' {  
  username: '', execute: '/usr/bin/id -un'  
}
```

However, this syntax should be used with caution as it could cause your program to run slowly if the command takes a long time to execute. Also, if you run a command which results in a very large output then this could cause NetConnect to consume a large amount of memory.

#### 4.3.4. Inheritance

To assist in removing duplication from configuration files, as usually the only main differences between profiles is the login credentials, profiles and proxies support the 'inherit' attribute. This attribute allows you to copy all the attributes (except nodes) so you don't have to redefine them:

```
profile: 'EX1' {  
  username: 'bloggsj'  
  password: 'Y2hyaXNAbmV0bm14Lm9yZw==', encoding: 'base64'  
  enable_level: '15'  
  run: 'term mon', 'term wid 0'  
}  
  
profile: 'EX2' {  
  inherit: 'EX1'  
}  
  
profile: 'EX3' {  
  inherit: 'EX1', except: 'enable_level', 'run'  
}  
  
profile: 'EX4' {  
  inherit: 'EX1'  
  enable_level: '9'  
}
```

In the above example we have defined our main profile 'EX1' that we are inheriting from in profiles 'EX2', 'EX3' and 'EX4'. You have the option of specifying the optional 'except' attribute which allows you to specify attributes you don't want to copy. To overwrite a copied attribute you just redefine it as in 'EX4'.

You aren't permitted to inherit from a profile that is already inheriting from another profile. Inheritance is supported for both profiles and proxies, but you can only inherit from the same type (i.e. a profile can't inherit from a proxy and vice versa).

#### 4.3.5. Node Types

Without any specific configuration under a profile, NetConnect will use the generic prompts associated with a Cisco device. This is fairly generic and will support a range of devices by other vendors including, but not limited to: Cisco, Juniper, Redback and Foundry. All of these nodes have similar prompts which the generic 'Cisco' node type works on.

Under the profile you may either specify a node type using the 'nodetype' attribute which loads some hardcoded prompts or you can specify a custom 'prompt' and 'expect' attribute. These two attributes are covered in more detail below. You may not specify the 'nodetype' attribute as well as the 'prompt' attribute under the same profile.

The following built in node types are currently supported:

**cisco** This is the default node type that will be associated with a profile if another node type isn't specified. This will also be assumed if the user hasn't specified a 'prompt' attribute – if a 'prompt' attribute is used then a node type of 'custom' is set. This node type works on a range of different vendors and allows the 'enable\_level' and 'enable\_password' attributes to be also defined under the profile.

If the 'enable\_level' attribute is specified then NetConnect will attempt to enter enable mode. When the node prompts for an enable password then NetConnect will use the value in 'enable\_password' if defined or will use the regular 'password' attribute.

**unix** This node type defines the prompts associated with a UNIX host and allows the 'su\_password' attribute to be specified. If this attribute is specified then NetConnect will issue the 'su -' command and use the password specified.

This feature also adds the 'Type: xx' syntax to the node output report so you can see what node type a specific profile is.

### 4.4. Configuration Schemas

The configuration schemas define what is allowed to exist within the configuration file and the format of these entries. For the purpose of the schema, square brackets are used to denote optional values.

#### 4.4.1. Profile Schema

A profile defines a template of information which can be reused when connecting to devices. Typically you would define login credentials within a profile which NetConnect can then use when logging into devices.

```

profile: '<id>' {
  inherit: '<value>'[, except: '<attrib>', ...]
  username: '<value>'
  password: '<value>'
  timeout: '<value>'
  break_sequence: '<value>'
  interactive_key: '<value>'

  include: '<value>'[, '<value>', ...]
  run: '<value>'[, '<value>', ...]
  proxy: '<value>'[, '<value>', ...]

  [
    nodetype: 'cisco'
    enable_level: '<value>'
    enable_password: '<value>'
  ]
  or
  [
    nodetype: 'unix'
    su_password: '<value>'
  ]
  or
  [
    prompt: '<value>'
    expect {
      match: '<value>', send: '<value>'
      ...
    }
  ]
  telnet[: '<group>'] {
    id: '<address>'[, ...]
    ...
  }
  ssh[: '<group>'] {
    id: '<address>'[, ...]
    ...
  }
}

```

Attributes in **red** below are mandatory and must be included for every profile defined.

- id** This is a unique identifier for the profile. This identifier will be used to reference this profile when using the '-p' command argument. You may only use alphanumeric characters, underscores and the hyphen within its name.
- inherit** This attribute allows you to specify the 'id' of another profile to inherit its attributes. Please see section 4.3.4 for information on profile inheritance.
- username** This is the login username that will be used for devices within this profile.

- password** This is the login password that will be used for devices within this profile.
- nodetype** This allows you to use one of the built in node types. Please see section 4.3.5 for more information on this attribute.
- timeout** This defines the amount of time that Expect will wait (in seconds) without receiving a response from the host before declaring a timeout. NetConnect is currently programmed to timeout after 10 seconds, but this can be tweaked using the 'timeout' attribute. This attribute is specified on a per profile basis so you can group devices which require a longer timeout together.
- break\_sequence** When you are executing scripts on nodes you can press CTRL+C to send a break sequence to NetConnect. Depending on the type of node you are connected to, determines how NetConnect processes it.

If you have defined a custom 'prompt' attribute then NetConnect will assume that you aren't connected to a Cisco device and will send CTRL+C to the node. If it thinks you are connected to a Cisco device it will send CTRL+^ which is the standard break sequence.

With this attribute you have the option to set a custom sequence on a per profile basis regardless of what NetConnect thinks the device is. An example is included below to demonstrate how to simulate a CTRL sequence:

```
profile: 'Example' {
  break_sequence: '\cC'
}
```

In the above example NetConnect will send CTRL+C to the node when the user presses CTRL+C during script execution. In the event you wish to disable this behaviour then you can pass through an empty string which disables this functionality.

- interactive\_key** When you have connected to a node you can press CTRL+D to enter interactive mode which allows you to run scripts. On certain nodes (i.e. Unix platforms) this might interfere with CTRL+D. This option allows you to change the key sequence for interactive mode to something which doesn't conflict.

If however you wish to disable the use of interactive mode then you can set this to an empty string which disables interactive mode.

```
profile: 'Example 1' {
  interactive_key: ''
}
```

```
}  
profile: 'Example 2' {  
  interactive_key: '\cR'  
}
```

In the above example in 'Example 1' we set 'interactive\_key' to an empty value which will disable interactive mode for all nodes under that profile. In 'Example 2' we remap the key sequence to CTRL+R so that the CTRL+D sequence will be passed through to the actual node.

### **include**

The 'include' attribute is used to import different configuration files into the main configuration file. This could be because of a split configuration design or to allow a global node list to be shared between all users. This attribute expects a single value or a list of values to denote multiple files. This attribute is detailed within section 4.5.

### **run**

This defines commands which you want NetConnect to perform before handing control back to the user. This would typically be commands that affect the terminal, etc. An example is shown below:

```
profile: 'Example' {  
  run: 'terminal monitor', 'terminal width 0'  
}
```

### **proxy**

In situations where devices reside behind SSH type proxy servers, NetConnect supports connecting through intermediate devices before attempting to connect to the device.

The 'proxy' attribute allows you to reference proxies that have been defined within the configuration file. You may specify multiple proxies separated by a comma if you wish to go through multiple proxies. NetConnect will then login to the proxies first before attempting to connect to the device.

### **prompt**

Although NetConnect supports Cisco routers and other vendors who use a similar prompt, this option allows you to define a custom prompt using a regular expression. If you have a piece of equipment that doesn't use the same prompt characteristics then you can define a custom regular expression which can be used. This is what NetConnect uses to know it has successfully connected to the device.

```
profile: 'Example' {  
  prompt: '^[\^$#]*(\|#)\s*$'  
}
```

The above example will match a standard UNIX prompt which is terminated with a '\$' sign and optional space afterwards. The square brackets are used to ensure it doesn't match a '\$' sign before it otherwise it could match on a comment.

## expect

The 'expect' attribute adds another level of flexibility or maybe complexity to the 'prompt' attribute. If you wish to use the 'expect' attribute then you must have defined a 'prompt' attribute. The 'expect' attribute allows NetConnect to send an arbitrary string if it sees a certain output before seeing the 'prompt' attribute.

To help explain this a bit better let's assume we are logging onto a UNIX host and we wish to 'su' to root (this is the same principal of what the 'unix' node type does). On UNIX hosts the character at the end of your prompt changes depending on the privilege (very much like a Cisco router). When you initially logon you will have a '\$' sign at the end of your prompt, but as soon as you 'su' to root the prompt will change to have a '#' sign at the end.

**Note:** This example shouldn't be used and the 'unix' node type should be used in preference to this example – this example is to explain the logic.

As we wish to 'su' to root we would define the prompt with the '#' sign at the end so NetConnect will only know it is connected when it sees this:

```
profile: 'Example' {  
  prompt: '^([^#]+)#\s*$'  
}
```

Now, this will normally fail as you won't see this prompt when you initially login and NetConnect will exit with a connection timed out error. This is where the 'expect' attribute comes in:

```
profile: 'Example' {  
  prompt: '^(?:[^\#]+#[^\$]+\$)\s*$'  
  expect {  
    match: '^([^\$]+\$)\s*$', send: 'su -'  
    match: 'Password:', send: '<PASSWORD>'  
  }  
}
```

In the above example, when NetConnect logs into a device, if any of the 'match' attributes match the output then NetConnect will send the text which is defined under the 'send' attribute. So, when NetConnect logs into a UNIX

host the 'prompt' won't initially match, but the first 'match' attribute does so it will send 'su -'. This will present the user with the 'Password:' prompt which will then match the second 'match' attribute and NetConnect will send the password. If the password is correct then we will be presented with a prompt with the '#' sign at the end which will match the 'prompt' attribute.

You may use the 'encoding' or 'ask' attribute on this line as well which will allow you to place an encoded or prompted value into the 'send' attribute.

To ensure we don't get ourselves into a loop there is a restriction that a match statement may only match once – it is removed from the match criteria once it has successfully matched.

## telnet

These are devices which use telnet as their access method. You may specify an optional group which is assigned to the devices after the telnet identifier. You should define the device and address using a 'key: value' pair with any extra attributes afterwards (these can be referenced from scripts). An example is specified below:

```
profile: 'Example' {
  telnet: 'Group' {
    R01: '192.0.2.1', vendor: 'Cisco'
    R02: '192.0.2.2', vendor: 'Juniper'
  }
}
```

The device should be a unique identifier as it will be used when attempting to match devices based on regular expressions. The address is the IP address or hostname of the device with an optional port specified using the syntax: 'address:port'.

## ssh

These are devices which use ssh as their access method. To save chopping down more trees you can reference the above section on telnet.

### 4.4.2. Proxy Schema

A proxy defines intermediate devices which should be connected to before NetConnect is able to connect to a device.

```
proxy: '<id>' {
  inherit: '<value>',[, except: '<attrib>', ...]
  username: '<value>'
  password: '<value>'
  method: '<value>'
  prompt: '<value>'
}
```

Attributes in **red** below are mandatory and must be included for every proxy defined (basically all of them).

**id** This is a unique identifier for the proxy. This identifier will be used to reference this proxy when using the 'proxy' attribute within a profile. You may only use alphanumerical characters, underscores and the hyphen within its name.

**inherit** This attribute allows you to specify the 'id' of another proxy to inherit its attributes. Please see section 4.3.4 for information on proxy inheritance.

**username** This is the login username that will be used for the proxy.

**password** This is the login password that will be used for the proxy.

**method** This attribute is used to define the command line which needs to be executed to connect to the proxy from the current device.

If you need to telnet to the proxy then the following is an example of the method which could be used:

```
proxy: 'Example' {  
  method: 'telnet 192.0.2.1'  
}
```

For SSH, you need to be aware that the username has to be specified on the command line unless you are using the same username as used locally:

```
proxy: 'Example' {  
  method: 'ssh -lbloggsj 192.0.2.1'  
}
```

In the above scenario the 'username' attribute would never be used, but is still mandatory.

**prompt** This is a regular expression that NetConnect uses to know it has successfully connected to the proxy.

```
proxy:  
  prompt: '^[^\$]+\s*$'
```

The above example will match a standard UNIX prompt which is terminated

with a '\$' sign and optional space afterwards.

### 4.4.3. Options Schema

The options schema is where the user may specify global NetConnect options which control the general use of NetConnect.

```
options {
  log_dir: '<value>'
  aliases {
    <alias>: '<value>'
    ...
  }
}
```

#### log\_dir

This option allows the user to control where log files are written to if a relative or absolute path isn't specified on the command line. By default, NetConnect will write the logfile to the local directory if no path is specified, but with this option they will be written to a specified directory.

For example, if './output.log' was specified then the logfile would be written to the local directory as you have specified path information (a '/' is used), whereas if 'output.log' was specified then the logfile would be written to the directory contained within the 'log\_dir' attribute.

#### aliases

An alias is used in NetConnect interactive mode which is initiated by pressing CTRL+D when connected to a node. The concept of an alias is to provide a shorthand way of running a script. This concept will make a lot more sense when reading about interactive mode, but I will explain the syntax within this section.

```
options {
  aliases {
    monitor: 'run /etc/netc/monitor.ncs'
  }
}
```

Within the 'aliases' attribute we define 'key: value' attributes where the 'key' is the actual alias name. In simple terms when you execute 'monitor' from interactive mode then it will execute 'run /etc/netc/monitor.ncs'.

### 4.5. Split Configuration

NetConnect supports splitting the configuration across multiple configuration files. The main 'netc.conf' file contains all the profile information, whereas the included file will only contain devices.

This feature is mainly used to create a shared global node list which everyone can include, but the login credentials are localised within your main configuration file.

#### 4.5.1. Main Configuration

Within the main 'netc.conf' configuration file we can use the 'include' attribute where we specify an additional configuration filename. You may specify multiple values for this attribute using a comma separated list.

```
profile: 'Example' {
  include: '/etc/netc/conf/global.conf'
}
```

#### 4.5.2. Included Configuration

The included configuration file uses a slightly different format as we aren't defining profiles. All the devices will be defined at the root level under the corresponding connection method ('telnet' or 'ssh'). These will then be imported into the relevant section of the main configuration file.

```
telnet: 'CUSTOMER' {
  CE-XX-01: '192.0.2.11'
  CE-XX-02: '192.0.2.12'
}

ssh: 'IPSEC-HUB' {
  IPSEC-XX-01: '192.0.2.21'
  IPSEC-XX-02: '192.0.2.22'
}
```

#### 4.6. Configuration Example

The snippet below is an example of a complete NetConnect configuration file. It demonstrates what a configuration file should look like and hopefully helps draw a picture of how the above attributes are used.

```
profile: 'RADIUS' {
  username: '', execute: '/usr/bin/id -un'
  password: 'Y24Lm9yZw==', encoding: 'base64'
  nodetype: 'cisco'
  enable_level: '15'
  run: 'term mon', 'term width 0'
  telnet: 'CE' {
    CE-XX-01: '192.0.2.11', vendor: 'Cisco', model: 'CRS-1'
    CE-XX-02: '192.0.2.12', vendor: 'Cisco', model: 'CRS-1'
  }
}
```

```

profile: 'RSA-SECURID' {
  username: '', execute: '/usr/bin/id -un'
  password: '', ask: 'Enter Token: ', type: 'one-time'
  nodetype: 'cisco'
  proxy: 'secure-jump-host'
  ssh: 'IPSEC-HUB' {
    IPSEC-XX-01: '192.0.2.21'
    IPSEC-XX-02: '192.0.2.22'
  }
}

profile: 'UNIX' {
  username: 'bloggsj'
  password: 'Y24Lm9yZW==', encoding: 'base64'
  nodetype: 'unix'
  su_password: 'Y24Lm9yZW==', encoding: 'base64'
  ssh {
    puffy.localdomain: '192.0.2.1', os: 'OpenBSD'
  }
}

proxy: 'secure-jump-host' {
  username: 'not-used'
  password: 'Y24Lm9yZW==', encoding: 'base64'
  method: 'ssh -lbloggsj 192.0.2.99'
  prompt: '^[\^\\$]+\$\\s*$'
}

```

## 5. Interactive Mode

When NetConnect is connected to a device the user may press CTRL+D which will result in the user being presented with a '[local dir] netc>' prompt. This enters NetConnect interactive mode which, at the moment enables two functions. You may exit interactive mode by pressing <enter> with no command specified which returns you to the device prompt.

On some platforms (i.e. Unix) CTRL+D will conflict with the local meaning of CTRL+D (i.e. used to logout and also used in Vi for page down). In this scenario interactive mode can be disabled or the sequence can be changed using the 'interactive\_key' option under the profile.

**Note:** Commands don't need to be completed, for example 'r' can be used instead of typing in 'run'. You just need to ensure you specify enough of the command for it to be unique.

### 5.1. Automatic Logoff

The first of these functions is to automate the logging off of the device. When NetConnect connects to a device it is the responsibility of the user to exit from the device. If NetConnect connected through a series of proxies / intermediate devices to reach the end device then the user would have to logout from multiple devices to get to where they started. By pressing CTRL+D (or different sequence if

remapped using 'interactive\_key' option) within NetConnect interactive mode NetConnect will then proceed to exit from the device and all subsequent proxy servers.

For this to work you must have a blank command line (i.e. you can't be in the middle of typing a command otherwise when NetConnect sends 'exit' the command won't make any sense). If you press CTRL+D after you have exited from the device but are still on the proxy then NetConnect will be out of sequence and won't know which proxy it is attempting to logout from. This is important as NetConnect will use the proxy prompt to determine a successful logout.

## 5.2. Running Scripts

The second function of NetConnect interactive mode, which is the most powerful, is the ability to execute scripts while still connected to a device. Normally scripts are executed by specifying them on the command line using the '-x' option, which will logon to the device, run the script and then logoff. By running the script from the '[local dir] netc>' prompt you will still be connected to the device when it finishes (as long as a command doesn't timeout). This method could be used to create a custom show command which will manipulate the output before displaying it.

To run scripts from interactive mode you need to use the 'run' command which allows you to specify a script file afterwards. If the script isn't in the current directory then you will need to ensure you specify the full path. After specifying the script file name you can optionally specify script options if your script supports them. The following example demonstrates how to run a script from interactive mode while also passing through a couple of script options:

```
[/home/bloggsj] netc> run /etc/netc/save.ncs cmd='show run' file='output.txt'
```

## 5.3. Running Shell Commands

NetConnect interactive mode also supports running shell commands while still connected to the device. This allows scripts to be modified without exiting from the device or loading a new terminal. As well as the 'exec' function we also have support for the 'chdir' function which allows you to change directory so NetConnect can reference files without having to specify the full path:

```
[/home/bloggsj] netc> exec vi myscript.ncs  
or  
[/home/bloggsj] netc> chdir /home/bloggsj/scripts
```

**Note:** The title of your XTerm terminal will still display the device name to remind you that you are still connected to a device. This is important if you decide to run a shell like 'bash' whilst still connected to a device.

## 5.4. Aliases

Aliases are used to allow the user to run scripts easily without having to type the full path to a script when running scripts in interactive mode. Aliases are defined within the configuration file and are stored under the 'options' attribute. Let's take the script example in section 6.2.1 using the syntax in section 5.2 above which runs the script 'save.ncs' located in '/etc/netc/'.

When using the syntax above you can see it is quite a mouthful, but with aliases we can shorten that a bit. Let's assume for the purpose of this example that we are always going to want to save the output of the command 'show running-config'. We would start by creating an alias as follows:

```
options {
  aliases {
    savecfg: 'run /etc/netc/save.ncs cmd="show running-config"'
  }
}
```

Now, when we are in interactive mode, if we specify 'savecfg' on the command line then this will be replaced for exactly what is defined within the configuration file. So, to simplify our original command we can now write:

```
[/home/bloggsj] netc> savecfg file='output.txt'
```

There is a way to shorten this even further as NetConnect will treat aliases which end with an '=' sign in a special way. If we specify an option with an '=' sign at the end of an alias then it will expect the first element on the command line to be assigned to that option. To demonstrate this, let's take our previous example and modify it slightly:

```
options {
  aliases {
    savecfg: 'run /etc/netc/save.ncs cmd="show running-config" file='
  }
}
```

Now, with the above syntax the first parameter specified on the command line will be assigned to the 'file' option. So, the final syntax would be as follows:

```
[/home/bloggsj] netc> savecfg 'output.txt'
```

When using this syntax to assign a value directly to an option using the '<option>=' syntax at the end of an alias line in configuration file you also have one further optimisation. If you are only passing a single option (i.e. the above example) then you may omit the quotes on space delimited values. NetConnect will check there isn't an '=' character present which would imply multiple options which

would result in a syntax error. It will then enclose the value with single quotes ensuring to escape any single quotes which are found within the body of the string. This would make the following example valid:

```
[/home/bloggsj] netc> savecfg /var/log/my router config.txt
```

Multiple aliases can be defined within the configuration file one after another. They will be shown on the usage screen if you press the question mark '?' in interactive mode.

### 5.5. Prompt Semantics

Assuming that your terminal works as it should, the '[local dir] netc>' prompt supports a rich set of features with regards to handling input from the user. As with most prompts it supports persistent history so you can recall a command you typed a long time ago. It also supports the most commonly used Cisco shortcut keys to assist you with editing your command:

- !** A single exclamation mark '!' will output the persistent history buffer which contains all the commands that you have entered previously. Each entry is numbered which allows you to easily reference it. You may also use the up and down arrow keys to cycle through the history buffer sequentially.  
  
To reuse a command in your history buffer all you need to do is enter the reference number on its own and hit enter. This won't execute the command, but will populate your input prompt with the history entry allowing you to edit it before executing it.  
  
To support persistent history across devices your history buffer will be stored in '\$HOME/.netc\_history'. This file will only be readable by yourself and will have the UNIX permissions of 600 (unless you decide to change it).  
  
Duplicate commands aren't added to your history buffer to save duplication, but if you use a previously used command then it will be moved to the bottom of the buffer.
- ?** A question mark '?' will display the help screen which is a cut down version of this section. However, it will also display aliases which you have defined within the configuration file.
- TAB** A tab character will enable tab completion on commands and aliases. The only requirement is that the cursor is at the end of the line for this to work. If multiple entries could match then it will output a list of those entries.
- CTRL+A** This sequence will move the cursor to the beginning of the line.

<b>CTRL+E</b>	This sequence will move the cursor to the end of the line.
<b>CTRL+X</b>	This sequence will delete everything from the cursor to the beginning of the line.

## 6. Scripts

Scripts allow you to automate certain tasks when connected to devices – there are no limits to what you can do with a script. NetConnect can execute scripts in two different ways, the first is via the command line where NetConnect will connect to a device, execute a script then disconnect gracefully. The second method is from NetConnect interactive mode which is entered using CTRL+D. In interactive mode a script is executed while you are connected to the device and you are left connected when the script finishes (on the condition the script doesn't cause a timeout).

Although not enforced, scripts generally have a file extension of '.ncs' which stands for 'NetConnect Script'. For the Vim people here using syntax highlighting, the following can be added to your '~/.vimrc' file to enable syntax highlighting for '.ncs' files to be based on Perl:

```
au BufNewFile,BufRead *.ncs set filetype=perl
```

Scripts are written in pure Perl and interact with the device through the ScriptObject (sO). The ScriptObject (sO) allows you to query the running properties of NetConnect as well as execute commands on devices.

### 6.1. Script Object (sO)

Within scripts, interaction with the device is through the ScriptObject (sO) package. The following table describes the different methods that this object supports:

```
sO->Node ([property])
```

This method allows you to query properties of the device that you have defined within the configuration file. The 'property' parameter is optional and if omitted, the method will return the current 'Node ID' that the script is being executed on. This can be useful if you wish the script to save data to a file which you can identify because it uses the 'Node ID'.

If a non-existent 'property' is specified then the method will return 'undef'. In addition to the 'Node ID', the 'address' property can be queried using this method. You may create custom properties within the configuration file as NetConnect isn't strict on checking for values which shouldn't be there. The following snippet demonstrates this concept:

```
profile: 'RADIUS' {
```

```
telnet: 'CE' {  
  CE-XX-01: '192.0.2.11', vendor: 'CISCO'  
}  
}
```

In the above example we have created the 'vendor' property which NetConnect doesn't use, but will transparently pass through. This can then be queried using this method from within your script.

### **s0->Profile ([property])**

This method is identical to the above 'Node' method, except it works for profiles instead of nodes. The 'property' parameter is optional and if omitted, the method will return the current 'Profile ID' that is being used by the node that the script is being executed on.

If a non-existent 'property' is specified then the method will return undef. Generally, the profile schema (Section 4.3.1) will define what can be queried, but not all properties are accessible via this method as they are referencing two different data structures. At the moment trial and error will need to be used, but plain properties like 'username' and 'password' won't cause a problem.

Like the above 'Node' method you may create custom properties within the configuration file as NetConnect isn't strict on checking for values which shouldn't be there.

### **s0->Option (<option> | <option> => <value>)**

This is a multi purpose method that can be used to set or get options depending on the parameters you pass. If you only pass a single parameter then NetConnect will return the value of this option. If you pass two parameters then NetConnect will attempt to set the option to the value passed.

```
my $R = s0->Option ('retry_on_timeout' => 1);  
  
or  
  
my $R = s0->Option ('break');
```

This method is used for both script options and also internal NetConnect options with the latter taking priority (i.e. if you pass 'node\_count' as a script option then NetConnect will return the internal 'node\_count' property and your script option will be ignored).

In the get scenario, if NetConnect doesn't know about the option you have requested then it will return 'undef'. In the set scenario, you are only permitted to set certain internal NetConnect options and aren't permitted to modify user provided script options. Internal NetConnect options are split into two categories: 'read-only' and 'read-write'. When setting an option the return

code will be '1' on success and 'undef' on failure. The internal NetConnect options and their category as listed below:

#### Read Only Options:

##### **break**

This option will return true (1) or false (0) if the user has pressed CTRL+C. This can be used within scripts at strategic points to stop execution if a break has been pressed.

##### **node\_count**

This option will return the amount of nodes that NetConnect is processing in total. When NetConnect runs a script on multiple nodes (using the 'file:<node file>' or 'group:<regex>' syntax), it will spawn multiple parallel processes. If NetConnect is processing multiple nodes in parallel this might cause an issue with file locking with regards to output files, etc.

If this option returns a value greater than 1 then you know NetConnect could be processing nodes in parallel and you can then use locking mechanisms to ensure exclusive access to shared resources.

##### **retry\_count**

This option will return the amount of nodes that NetConnect has reconnected to the node. When NetConnect connects to the node for the first time this will return 0, but as soon as it is disconnected due to a timeout and 'retry\_on\_timeout' has been set then this value will increment. This is useful if you want to truncate a log file the first time you process a node.

##### **pid**

This option will return the current UNIX process ID of the parent NetConnect process. This method can be useful when creating unique files or when naming lock files as they will be specific to the current running process.

##### **verbose**

This option will return the current verbosity level. Within a script a verbosity level of 0 will mean that you won't see the output from commands and a verbosity level of 1 will mean you will.

Scripts will have a verbosity level of 0 by default, unless you have specified the '-v' parameter on the command line.

#### Read Write Options:

##### **retry\_on\_timeout**

This option allows you to tell NetConnect to reconnect to a node and run the script again if it times out. Setting a value of '1' will cause NetConnect to keep reconnecting to a node if it keeps on timing out. It will only reconnect on certain timeouts and needs to be able to connect to the node successfully before you can use this option.

For example, an invalid username or password won't cause NetConnect to reconnect to the node, but if a command was to timeout then NetConnect would.

Be aware if setting this option in a script and processing multiple nodes. NetConnect can only process a limited amount of nodes in parallel and if a script loops indefinitely then some nodes won't be processed.

```
s0->Run (  
  <command>,  
  [ Timeout => xx ],  
  [ Prompt => qr/xx/ ],  
  [ Callback => CODEREF ],  
  [ Match => REF ]  
)
```

This method can be invoked in a number of different ways to achieve slightly different results. The case-insensitive parameters 'Timeout' and 'Prompt' are optional and if omitted NetConnect will use previous values. If 'Timeout' is 'undef' then NetConnect will wait indefinitely for the command to finish. The 'Prompt' parameter expects a compiled regular expression which needs to be passed using the 'qr//' operator.

The 'command' parameter can be passed in two different ways; 1) it can be passed as a text string with the actual command, i.e. 'show running-config' or 2) by passing a reference to an array which contains a list of commands:

```
my @ncc = (  
  'terminal length 0',  
  'show version',  
  'show running-config'  
);  
  
s0->Run (\@ncc);
```

To store the output from the command the 'Run' method can return this in a number of different ways. The return value is context dependent and is based on the required lvalue. If the lvalue is a list (@) then the return value will be a list, whereas if the lvalue is a scalar (\$) then the return value will be a scalar with each line separated by a "\n".

If you pass multiple commands using a referenced array then the 'Run' method will return 'undef' in scalar context and an empty list in list context. If you want the output saved from the commands then you will need to pass the commands individually to the 'Run' method. This is the same of the 'Callback' and 'Match' parameters.

**Note:** If the user presses CTRL+C to interrupt the script then the return value of the method will be 'undef' in a scalar context and an empty list in list context.

### Callback Parameter

The 'Callback' parameter is useful if you wish to modify the output of commands inline before the output is sent back to the user. If you specify a code reference (e.g. '\&sub') then for each line that is sent back from the device it will be passed to the function specified as the first parameter.

By using the 'Callback' parameter it will disable the output during the command – if you want to see the output then your callback function will need to output the line. This will be reset to what it was before when it stops running the command.

### Match Parameter

The 'Match' parameter is used to return the actual string that was matched within the prompt. This can be useful to determine if the device requested a password, etc. This should be passed by reference (e.g. '\\$vMatch').

Apart from the above syntax with regards to the ScriptObject (sO), the rest of a script is written in Perl. You can use this to manipulate the results or to run commands based on different outputs.

## 6.2. Examples

The following section looks into scripts in a bit more detail and gives some examples of how they can be used.

**Note:** None of the below example scripts contain any error checking for simplicity – they should not be used without adding some basic error checking.

Perl can be a very cryptic language to understand depending on how a Perl script is written. It is commonly used in obfuscation competitions due to its cryptic nature. Some syntax also has a very compact notation which doesn't make much sense to people who aren't familiar with it. However, Perl can also be very simple and for the purpose of NetConnect scripts works well as it provides a very powerful regular expression library to manipulate outputs.

### 6.2.1. Save Script

This script can be used to save the output of a command to a local file. Although we have the ability to save the session output to a logfile using the '-l' parameter, we might want to just save a specific command and not the entire session.

This script uses two script options 'cmd' and 'file', neither of which are optional. The 'cmd' option is used to pass through the command to run and the 'file' option is used to specify the output file.

```
if (sO->Option ('node_count') == 1) {
```

```

my $T = sO->Run ('show terminal | include ^Length:');
sO->Run ('terminal length 0');

my $O = sO->Run (sO->Option ('cmd'));
if (open (my $fh, '>', sO->Option ('file'))) {
    print $fh $O;
    close ($fh);
}

if ($T =~ m/^Length: (\d+)/) {
    sO->Run ('terminal length ' . $1);
}
}

```

So, we are basically checking that we are running this script on a single device otherwise we would have issues with the output file being overridden by subsequent nodes. Once we are running on a single node we retrieve the current terminal length so we can restore it after we have finished. We then run the command (script option 'cmd') saving the output to a variable which we then write to the output file (script option 'file'). Once we have saved the output we restore the terminal length.

To run this script from the NetConnect command line, assuming it is called 'save.ncs' we would use the following syntax:

```
$ netc <node> -x save.ncs -o cmd='show running-config' -o file='output.txt'
```

To run this script from NetConnect interactive mode when we are already connected to the node we would use the following syntax (after pressing CTRL+D):

```
[/home/bloggsj] netc> run save.ncs cmd='show running-config' file='output.txt'
```

## 6.2.2. Inventory Script

This script is more suited for running on multiple devices at the same time as it creates a CSV file which contains all the hardware items with serial numbers that are found on a Cisco ISR router.

This script uses the mandatory script option 'file' which specifies a file to write the inventory information to. This file will be appended with new entries.

```

my $T = sO->Run ('show terminal | include ^Length:');
sO->Run ('terminal length 0');

my @O = sO->Run ('show inventory');
if (sysopen (my $fh, sO->Option ('file'), O_WRONLY|O_CREAT|O_APPEND, 0600)) {
    flock ($fh, LOCK_EX);
    foreach (@O) {

```

```

    if (m/^PID:\s+(\S+).+SN:\s+(\S+)$/i) {
        print $fh sO->Node . ',' . $1 . ',' . $2 . "\n";
    }
}
close ($fh);
}

if ($T =~ m/^Length: (\d+)/) {
    sO->Run ('terminal length ' . $1);
}
}

```

As before in the previous script, we take a copy of the terminal length so we can restore it once we have finished. We run the command 'show inventory' and process the output to obtain the PID and SN for each item. Once we have that information we write it to the file specified using the 'file' script option. We use the 'flock' function to ensure we have an exclusive lock on the file – if processing multiple nodes at once then multiple processes could try writing to the same file at the same time.

To run this script from the NetConnect command line, assuming it is called 'inventory.ncs' and we have a file called 'nodes.txt' which contains a list of nodes we want to run it on, we would use the following syntax:

```
$ netc file:nodes.txt -x inventory.ncs -o file='output.txt'
```

To run this script from NetConnect interactive mode when we are already connected to the node we would use the following syntax (after pressing CTRL+D):

```
[/home/blogsj] netc> run inventory.ncs file='output.txt'
```

### 6.2.3. Monitor Script

The monitor script is used to run the same command every second and output the result onto the same place on the screen. This is probably the most complicated script out of the examples, but the output is pretty useful so I couldn't resist but including it. If you are familiar with the UNIX 'top' command then this inhibits the same behaviour, but without the limitation of only show processes.

The script uses a single option 'cmd' which isn't optional and defines what command NetConnect should execute every second. This script also defines a 'cdiff' subroutine which is used to colour the differences between the outputs (like the linux 'watch' command).

```

if (sO->Option ('node_count') == 1) {
    if (defined sO->Option ('cmd')) {
        my $T = sO->Run ('show terminal | include ^Length:');
        sO->Run ('terminal length 0');
    }
}

```

```

my $rFlag = 1;
local $SIG{'INT'} = sub {
    $rFlag = 0;
};

my @O = ();
while ($rFlag) {
    my @N = sO->Run (sO->Option ('cmd'));
    my $T = &cdiff (\@O, \@N);
    print "\e" . '[2J' . "\e" . '[1;1H';
    print $$T . "\n";
    print '<CTRL+C to STOP>';
    sleep (1);
    @O = @N;
}

if ($T =~ m/^\Length: (\d+)/) {
    sO->Run ('terminal length ' . $1);
}
}

sub cdiff {
    my ($O, $N) = @_;
    my $T = '';

    for (my $i = 0; $i < $#{$N} + 1; $i++) {
        if (defined @{$O}) {
            if ($i <= $#{$O}) {
                if ($O->[$i] ne $N->[$i]) {
                    my @tO = split (m//, $O->[$i]);
                    my @tN = split (m//, $N->[$i]);

                    for (my $j = 0; $j < $#tN + 1; $j++) {
                        if ((($j > $#tO) or ($tN[$j] ne $tO[$j])) {
                            $T .= "\e" . '[1;37;41m' . $tN[$j] . "\e" . '[0m';
                        }
                        else {
                            $T .= $tN[$j];
                        }
                    }
                    $T .= "\n";
                }
                else {
                    $T .= $O->[$i] . "\n";
                }
            }
            else {
                $T .= "\e" . '[1;37;41m' . $N->[$i] . "\e" . '[0m' . "\n";
            }
        }
        else {
            $T .= $N->[$i] . "\n";
        }
    }
}
}

```

```
return (\$T);  
}
```

So, we are basically checking that we are running this script on a single device otherwise we would have issues with the screen output being overridden by subsequent nodes. Once we are running on a single node we retrieve the current terminal length so we can restore it after we have finished. We then start a loop which obtains the output of your command 'cmd' and outputs it to the screen after clearing the screen.

This script would typically be executed from interactive mode to monitor interfaces or processes, but could be executed from the command line. We will assume the script is called 'monitor.ncs' and that we wish to create an updating 'top' style output on a Cisco router for the purpose of this example:

```
$ netc <node> -x monitor.ncs -o cmd='show processes cpu | exclude 0.00% +0.00% +0.00%'
```

To run this script from NetConnect interactive mode when we are already connected to the node we would use the following syntax (after pressing CTRL+D):

```
[/home/bloggsj] netc> run monitor.ncs cmd='show proc cpu | exc 0.00% +0.00% +0.00%'
```