Scenario Commands

Scenario Commands can be called upon in scenarios. They can return nothing, a list variable or a hash variable. When they fail, the <error>-variable is set.

Several of the commands will benefit from the support of 'here documents' as a way to avoid creating trivial templates. Please see the section on Here documents

Commands

cmd_exec

Execute command line-by line on cli of a NetYCE managed node.

Mandatory options:

```
    -n node         hostname of the node
    -f cmd_file     file containing the commands you want to execute
```

Optional options:

```
    [-o <origin>]    use the 'origin' node as the context node. Commands
are executed on the -n node.
    [-c <connect>]   use 'console', 'api' or 'management' connection
    [-a <address>]   overrides the address of the node
    [-d <domain>]    overrides the management domain of the node
    [-v <vendor>]    overrides the vendor type of the node
    [-q]             sets quick-mode: skip config save/backup/nccm
    [-e]             cancels the normal stop-on-error behaviour
    [-u <operator>]  use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

The `-e` option overrides the default behaviour of aborting all remaining commands when an error was detected. When used, the device response is not checked for errors or warnings and execution proceeds with the next command. The logs will not indicate an error was encountered.

cmd_exec_basic

Execute command line by line on cli of a node in the CMDB table

Mandatory options:

```
    -n node             hostname of the node
    -f cmd_file         file containing the commands you want to execute
    -a <node_addr>      the address of the node
    -d <domain>         the management domain of the node
    -v <vendor_type>    the vendor type of the node
```

Optional options:

```
    [-c <connect>]   use 'console', 'api' or 'management' connection
    [-u <operator>]  use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

resched_job

> Note This command is not available for releases prior to 7.2.0.

Schedule the running job to execute again. The `resched_job` command allows for recurring jobs. It is included in a scenario to reschedule the same job for a later day and/or time. Since the rescheduled job will execute the same scenario, this job will again be rescheduled for a later moment. In effect, the task will become a repeated job.

To stop the job from rescheduling itself it needs to be manually 'cancelled' using the “Operate - Jobs” tool.

Mandatory options

It is mandatory to specify the `-i` (interval) or `-t` (daytime) option.

```
    -i interval     reschedule this job at the "interval" after current
start time
                    "interval" format in 'd' 'h' 'm' or a combination
like "1d 2h 30m' or '72h"
                    no 'now' supported
    -t daytime      reschedule this job at "daytime"
                    "daytime" format like "tomorrow 5:05" or "sunday
0:00"
                    no 'now' or date supported
```

A maximum of eight (8) days can be scheduled ahead of the current time using either the 'interval' or 'daytime' option. The valid values for the 'day' in the “daytime” format are: today, tomorrow, monday, tuesday, wednesday, thursday, friday, saturday, sunday, or their three-letter abbreviations.

Using a '-t' option like 'today 18:00' will result in a job that repeats itself at 18:00 the same day, but since at that time this 18:00 will be past, the re-scheduling will stop. No error is raised for this. If the '-t' is 'tomorrow 18:00' the rescheduled job will again be scheduled for the next day as its targeted time will forever be in the future.

Optional

```
    [-s]  set server selection to '-auto-' instead of using same server
```

Job parameters are reused from the scheduler and job files: - reuse scenario file - reuse command file - reuse parameters file - operator will stay the same - reuse the change-id - reuse the queue - reuse the description - jobtype stays the same - reuse job-rule validations / approvals

The NetYCE server where the job will be rescheduled is the same as the current server by default. Only when the '-s' option is used will the server set to `-auto-`. It is not possible to assign a different server directly.

Configs

config_create

Generate commands from template.

Mandatory options:

```
    -n node                         hostname of the node
```

Optional options:

```
    [-t <template>]                 overrides the default main-template of
the node
    [-f <out_file>]                 override the default output file
'<node>.cmd'
    [-p parameter=<value>]          define or overwrite variables for use in
template. it will not overwrite parameters
                                    It should be quoted if the value has
whitespaces.
    [-i <interface-list>]           generate the config(s) for the portnames
listed
    [-x]                            extend (append) the commands to the
output file if one exists
```

Parameters already known within the context are not required to be provided manually.

Multiple '-p' arguments may be provided.

Configuration can also be created 'inline' using the following syntax. And pushing the configuration using cmd_exec:

```
config_create -n <node> -f <node>_ntp.cmd <<EOT
  ntp server <ntp_server1>
  |<ntp_server2>|ntp server <ntp_server2>
EOT
cmd_exec -n <node> -f <node>_ntp.cmd
```

The here documents describe this functionality in more detail.

config_startup

Upload full config file and make startup.

Mandatory options:

```
    -n node          hostname of the node
    -f cfg_file      file containing the config you want to upload
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
    [-u <operator>] use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

config_save

Save config on device and download to nccm.

Mandatory options:

```
    -n node          hostname of the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
    [-m <message>]  adds a message to the nccm as the cause
    [-u <operator>] use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

config_diff

Fetch running config and save it into the nccm

Mandatory options:

```
    -n node         hostname of the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
```

This command will return an error if its backup cannot be retrieved or saved.

config_restore

Upload config from NCCM to the node as startup-configuration.

Mandatory options:

```
    -n node         hostname of the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
    [-u <operator>] use NetYCE <operator> credentials for first attempt
to logon to the device.
    [-s <select>]   specifies the NCCM configuration to select
                    may be one of:
                    'previous'  select most recent config prior to a
command job.
                                it selects the latest poll or pre-config
backup available.
```

```
                                this is the default action if no -s
option is provided
                    'last'      select the most recent NCCM backup
available
                    'poll'      select the most recent NCCM polled backup
                    'marked'    select the NCCM config manually
identified ('marked')
                                using the NCCM 'Config diff' tools.
                                the mark will NOT be cleared after
execution.
                    <jobid>     select the NCCM config created by
specified jobid.
                                jobs can create a pre-config backup if a
config change was detected
                                before a config change is made, but will
also create a post-config backup.
                                the config selected will be the final
(post-config) backup the job executed.
                    <polltime>  select the (first available) NCCM config
specified by date and timestamp.
                                the date (yyy-mm-dd) must included in
full, the time may be partial (hh:mm:ss).
                                a valid option value is therefore
"2018-04-25 14:28".
                    <nccmid>    select the NCCM config using the
specified nccmid.
                                this option is only useful when direct
database access is available.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

Parsing

parse_cmd

Run a show command on a node, and parse its output. Returns a hash variable with its result, trying to put the results in a regular list variable will fail. (<%variable>, instead of <variable>).

Mandatory options:

```
    -n node          hostname of the node
    -t <template>    a command parsing template. More information on
[[howto:command_parsing_templates|command parsing templates]]
    -r <request>     a request which command to execute to the node, for
```

```
example "show interface brief"
    -c <client_type> the client type of the node. Use this if you have
multiple nodes with the same hostname (for example in the CMDB and YCE
database)
```

Some vendors allow the option to specify pipes '|' in the request. This is also allowed to be passed through the command parsing, though you do need to be careful if you wish to use multiple pipes.

An example is shown for Junos, which allows multiple pipes with multiple commands. In this case it is required to have them escaped (escaped with '\').

> Before 7.1.0 the syntax was <@cmd>

```
<%cmd> := Parse_cmd -n <node> -t parsing_template -r "show configuration
routing-instances <Vrf_name> \| display set \| match interface"
```

Device

reboot_node

Restart the device.

Mandatory options:

```
    -n node         hostname of the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
    [-u <operator>] use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

clear_console

Reset the console line on the terminal server

Mandatory options:

```
    -n node          hostname of the node
```

Optional options:

```
    [-c <connect>]   use 'console', 'api' or 'management' connection
    [-a <address>]   overrides the address of the node
    [-d <domain>]    overrides the management domain of the node
    [-v <vendor>]    overrides the vendor type of the node
    [-t <term_srv>]  terminal server name
    [-l <line>]      line number on terminal server to reset
    [-u <operator>]  use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

File transfer

file_get

Transfer a file TO the named node. The transfer uses SFTP or TFTP depending on the available support from the vendor and the connectivity available.

Mandatory options:

```
    -n node         hostname of the node
    -s source       location and name of the existing file on the NetYCE
system
    -t target       location and name of the new file on the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
```

file_put

Transfer a file TO the NetYCE system. The transfer uses SFTP or TFTP depending on the available support from the vendor and the connectivity available.

Mandatory options:

```
    -n node         hostname of the node
    -s source       location and name of the existing file on the node
    -t target       location and name of the new file on the NetYCE
system
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-v <vendor>]   overrides the vendor type of the node
    [-u <operator>] use NetYCE <operator> credentials for first attempt
to logon to the device.
```

The `-u` option refers to a NetYCE user account. The username and password of this account will be used to login to the device. When available, the `Device password` value will be used as the password instead of the login password of the account. This NetYCE account will be used as the first set of credentials to login to the device. The `rme_user` and `local_user` credentials will still be attempted should login fail.

OS Upgrades

Also see page on OS-repo scenario calls

os_files

retrieves a list of os files from the OS repository. Returns an error if no os files are found for these criteria.

Optional options:

```
    [-n <name>]        the name of the os image",
    [-v <vendor>]      the vendor type of the os image",
    [-s <status>]      the status of the os image (production, historic,
planned, unknown; default is production)",
    [-t <device_type>] the device type of the os image",
    [-f <version>]     the version of the os image",
    [-d <domain>]      the node type of the os image",
    [-y <node_type>]   overrides the node type of the os image",
    [-o <file_type>]   the file type of the os files: (os-image, boot-
image, license, other, unknown; default is os-image)",
```

os_image_select

retrieve os repo image data. The returned hash variable has the following attributes: Id, Error, Status, Vendor_type, Device_type, Os_name, Os_version, Os_path, Domain, Node_type, Pre_activation_cmd, Post_activation_cmd.

If the options result multiple os images, the image with status production will be returned. If there are more than that, the one with the latest timestamp will be returned. If you want to guarantee that no more than one os image will be selected, then provide a filter for vendor type and device type, since only one os image in production status can exist for each vendor type/device type combination,

Optional options:

```
    [-n <name>]        the name of the os image
    [-v <vendor>]      the vendor type of the os image
    [-s <status>]      the status of the os image (production, historic,
planned, unknown; default is production)
    [-t <device_type>] the device type of the os image
    [-f <version>]     the version of the os image
    [-d <domain>]      the domain of the os image
    [-y <node_type>]   the node type of the os image
```

os_file_select

retrieve the files of an os repo image as a list of hash variables. Each hash variable has the following attributes: Os_file_name, Vendor_type, Type, Status, File_size, File_date, File_md5, File_crc, Storage_min

If the options result multiple os files, the one with the newest timestamp will be returned.

Optional options:

```
    [-i <id>]          the os image id the files belong to
    [-v <vendor_type>] the vendor type of the file
    [-n <name>]        the name of the file
    [-t <type>]        the file type of the file (os-image, boot-image,
license, other, unknown). Default is os-image
```

DNS

dns_add_host

This command is available only with the Infoblox integration operational.

Optional options:

```
    [-n <node>]     hostname of the node to add to DNS.
                       Mandatory if '-f' is not used
```

```
                    Do not use the fqdn here.
    [-f <fqdn>]        the full qualified host name.
                         Mandatory if '-n' is not used.
                         Overrides any associated with '-n'.
    [-a <ipv4>]        the ipv4-address of the DNS record. Overrides any
associated with '-n'.
                         The ipv4-address must include a /prefix in the
format 'address/prefix'.
                         Use a /32 prefix to bypass the subnet-size
check.
    [-6 <ipv6>         an optional ipv6-address for 'host' records only.
Cannot be used without an ipv4 address.
                         The ipv6-address must include a /prefix. Use
prefix /128 to bypass subnet-size test.
    [-t <type>]        dns record type: 'host' or 'arec'.
                         Default to 'host'.
    [-v <ipam_view>]   specify the Infoblox IPAM-view. The default is
configured in 'etc/<server>_dhcp.conf'
                         which is normally 'Default'.
    [-d <dns_view>]    specify the Infoblox DNS-view. The default is
configured in 'etc/<server>_dhcp.conf'
                         which is normally 'Intern'. The DNS-view is
hierarchically dependent on the IPAM-view
    [-r]               flag to renew the existing ip-address of the DNS
record.
                         Requires the '-a' option to provide the new
address
    [-i <string>] information string added to the DNS record.
```

The **`dns_add_host`** will create by default a DNS 'host' record for the node using its management ip-address. If the node exists in the YCE database, only the **`-n`** option is needed:

```
dns_add_host -n <node>
```

If the node is present in the CMDB table, the fqdn is retrieved from this table, but the ip-address must be included explicitly using the **`-a`** since it obviously cannot be resolved using the DNS.

```
 # requires subnet '10.1.2.16/29' to exist in IPAM
 dns_add_host -n <node> -a 10.1.2.17/29
```

It is mandatory to specify the ip-address with its subnet's prefix (1-32). The DNS policies in place require the IPAM subnet to exist or adding a DNS a-record or host is not permitted.

The only exception to this policy is where a /32 (single address) subnet is targeted. In that case, the address is tested to exist and is available (unused), but the subnets prefix needs not to be explicitly known.

```
 # only requires the address to exist in IPAM, but subnet-size is ignored
 dns_add_host -n <node> -a 10.1.2.17/32
```

The '-n' option can be replaced with the **`-f`** option to specify the full-qualified-domain-name (fqdn).

```
# create the DNS host using the fqdn
dns_add_host -f new-host.my.domain -a 10.1.2.17/29
```

If the fqdn is available in the YCE database, the management address of the node is located and used, but otherwise the address option is needed too. The `-f` option allows a different domain name to used than registered in the YCE database.

Policies demand that a domain name must pre-exist as a 'zone' in the IPAM or the request will be denied. This applies to the explicit domains given in the `-f` option, but also the implicit domains used with the `-n` option.

By default a 'host' type record will be created. Setting the **`-t`** option with the value 'arec' will create an A-record instead. The `-t` can be either 'host' or 'arec'. Both record types are checked when adding a new record to verify it does not already exist. The ERROR flag is set when either exists or when any of the many criteria and policies are not met.

The Infoblox environment requires the use of IPAM-views and DNS-views. The DNS-views are associated with the IPAM-views. Both views have defaults assigned in the 'etc/<server>_dhcp.conf' configuration file. Use the -v and -d options to override these defaults.

> When creating a host, the fqdn must be unique and not exist, the domain-name must exist in the DNS-view, and the subnet must exist in the IPAM-view. A 'host' record must have at least one ipv4-address assigned, regardless of any ipv6-addresses. The ipv4 address and ipv6 address must reside in the same IPAM-view.

```
# remove a host from the IPAM-view 'Internet' using the DNS-view 'Exposed'
dns_clear_host -n <node> -f "foo.acme.com" -t host -v Internet -d Exposed

# add a host to the IPAM-view 'Internet' using the DNS-view 'Exposed'
dns_add_host -n <node> -f "foo.acme.com" -t host -a 172.122.34.20/24 -6 1a04:bc8:3001::20/120 -v Internet -d Exposed
```

> Note: currently the ipv6 support is limited to 'host' records. An extension to support AAAA-records is planned.

For additional details, see the article on “Add_host” in Infoblox DNS API plugin. It is this API call that will be executed when using the dns_add_host command.

> These are the four new DNS commands and rely on the NetYCE - Infoblox integration. Please note that in these early-release DNS commands there is no support for the Infoblox Extended Attributes.

dns_clear_host

Remove DNS host. This command is available only with the Infoblox integration operational.

Optional options:

```
    [-n <node>]        hostname of the node to add to DNS.
                         Mandatory if '-f' is not used
                         Do not use the fqdn here.
    [-f <fqdn>]        the full qualified host name.
                         Mandatory if '-n' is not used.
                         Overrides any associated with '-n'.
    [-t <type>]        host or arec, overrides default host type record
    [-v <ipam_view>]   specify the Infoblox IPAM-view. The default is
configured in 'etc/<server>_dhcp.conf'
                         which is normally 'Default'.
    [-d <dns_view>]    specify the Infoblox DNS-view. The default is
configured in 'etc/<server>_dhcp.conf'
                         which is normally 'Intern'. The DNS-view is
hierarchically dependent on the IPAM-view
    [-c]               also remove cnames
```

dns_add_alias

Add DNS alias.
This command is available only with the Infoblox integration operational. Edit the configuration file `/opt/yce/etc/<servername>_dhcp.conf` to define GridMaster, policies and defaults.

Use the type ('-t') 'alias' to add an alias to a 'Host'-record or the type 'cname' to create a 'Cname'-record to point to a canonical host or a-record.

The use of the '-n' option implies the node name must exist in the YCE or CMDB environments of NetYCE. Otherwise the '-f' option should be used to provide the full-qualified-domain-name by which the host or canonical a-record record can be located in Infoblox. The node or fqdn cannot be ipv4 or ipv6 addresses.

Aliases should be included as a fqdn. However, should it look like a hostname, and a host domain is available, the alias will be extended with use the host domain to find it. Multiple aliases may be specified in one call.

Options:

```
    -n <node>        the hostname of the node in question, either the
hostname or the fqdn is required
    -f <fqdn>        the the fqdn of the node in question, either the
hostname or the fqdn is required
    -a <alias>       alias name(s) for <node> as fqdn or hostname (where
the <node> domain will be used)
    [-t <type>]      'alias' or 'cname', default is 'cname' type record
    [-d <dns_view>]  specify the Infoblox DNS view name - default set in
etc/<server>_dhcp.conf
    [-i <info>]      add a comment to the record
    [-e <ext_attr>]  define an extended attributes (use "param=value"
format)
```

dns_clear_alias

Remove DNS alias.
This command is available only with the Infoblox integration operational. Edit the configuration file `/opt/yce/etc/<servername>_dhcp.conf` to define GridMaster, policies and defaults.

The behaviour when removing an alias from a 'Host'-record is different from removing a 'Cname'-record that points to a canonical host- or a-record.

When type ('-t') is 'alias', the aliases ('-a') will be removed from an existing 'Host'-record. The use of node ('-n') or fqdn ('-f') option is then mandatory. The included aliases will only be removed if they belong to that host record.

If the type is 'cname', the use of the node ('-n') or fqdn ('-f') is optional. When included, this information is used to verify the canonical name matches the cname. If not, the cname is not removed. Omitting the node and fqdn will result in removing the cname without verification.

The use of the '-n' option implies the node name must exist in the YCE or CMDB environments of NetYCE. Otherwise the '-f' option should be used to provide the full-qualified-domain-name by which the host or canonical a-record record can be located in Infoblox. The node or fqdn cannot be ipv4 or ipv6 addresses.

Aliases should be included as a fqdn. However, should it look like a hostname, and a host domain is available, the alias will be extended with use the host domain to find it. Multiple aliases may be specified in one call.

Options:

```
    -n <node>        the hostname of the canonical node, either the hostname or the fqdn is required for host type
    -f <fqdn>        the fqdn of the canonical node, either the hostname or the fqdn is required for host type
    -a <alias>       alias name(s) for <node> as fqdn or hostname (where the <node> domain will be used)
    [-t <type>]      'alias' or 'cname', default is 'alias' type record
    [-d <dns_view>]  specify the Infoblox DNS view name - default set in etc/<server>_dhcp.conf
```

dns_clear_ip

This command is available only with the Infoblox integration operational.

The `dns_clear_ip` command will search the DNS for any Host or A-record referring to an ipv4-address and remove it. Any CNAME records pointing to these removed records will be removed as well.

Since an Infoblox Host construct can be associated with multiple ip-records, only the targeted ip-address is removed. Should such a Host constrict at that point have no ip-addresses left, then the entire Host is deleted. CNAMES pointing to Hosts constructs are preserved if ipv4-addresses remain.

Optional options:

```
[-a <ipv4>]  the ipv4-address of the DNS records to search for and remove
(without prefix).
               Overrides -n and -f option.
[-n <node>]  nodename of the node whose management ip-address is to be
removed.
               Do not use the fqdn here, the nodename is assumed to be a
known NetYCE object.
[-f <fqdn>]  full qualified host name of a DNS record to be removed using
its resolved ip-address.
               Only the first ip-address that the DNS returns is searched
for.
```

By default, the DNS view searched is 'Intern' (as defined in the NetYCE 'DNS Infoblox API' plugin).

Examples:

```
dns_clear_ip -a 10.106.16.10
dns_clear_ip -n ams-cr01
dns_clear_ip -f server-1234.acme.com
```

External calls

The external calls is a group of traps, signals and other exec calls.

trap_nms

The `trap_nms` command sends a SNMP trap to the NMS management stations. The management stations are specified using the `-a` option, one for each station.

The trap can include message details that require the `-e` option with the SNMP enterprise ID and a `-v` with the variable OID (object-id). Then by including a `-m` option with a message, this message gets assigned to the variable OID.

By including more `-m` messages, each is then assigned to the next OID by auto-incrementing the last digit of the OID. The order of the -m's determine which OID is used. The default -v is '1.0' which is used for the first -m, the second would then be 1.1 and so on. By defining your own -v <varoid> you have full control over the object-identifier.

Example:

```
[scenario]
Description <node> trap-test

trap_nms -a 172.17.10.21 -a 172.17.10.28 -e 1006 -v 3.2.1 -m
"hostname:<hostname>" -m "domain:<domain>" -m "somevar:someval"

```

```
end
```

The traps sent will in this case include a 'varbindlist' with the following information:

```
{
  '1.3.6.1.4.1.1006.3.2.1' => 'hostname:ACME-GV-023',
  '1.3.6.1.4.1.1006.3.2.2' => 'domain:pub.acme.com'
  '1.3.6.1.4.1.1006.3.2.3' => 'somevar:someval',
},
```

Mandatory options:

```
    -a <nms_addr>     send the trap to this address
```

Optional options:

```
    [-e <enterprise>] the enterprise OID to use (default 696)
    [-v <var_oid>]    the variable OID to use (default 1.0)
    [-m <message>]    the OID message text
    [-s <specific>]   the specific trap type to use (default 0)
```

trap_node

The `trap_node` command sends a SNMP trap to the NMS management stations but 'spoofs' the node as the originating address. The management stations are specified using the `-a` option, one for each station.

The trap can include message details that require the `-e` option with the SNMP enterprise ID and a `-v` with the variable OID (object-id). Then by including a `-m` option with a message, this message gets assigned to the variable OID.

By including more `-m` messages, each is then assigned to the next OID by auto-incrementing the last digit of the OID. The order of the -m's determine which OID is used. The default -v is '1.0' which is used for the first -m, the second would then be 1.1 and so on. By defining your own -v <varoid> you have full control over the object-identifier.

Example:

```
[scenario]
Description <node> trap-test

trap_node -n <node> -a 172.17.10.21 -e 1006 -v 3.2.1 -m
"hostname:<hostname>" -m "domain:<domain>" -m "somevar:someval"

end
```

The traps sent will in this case include a 'varbindlist' with the following information:

```
{
  '1.3.6.1.4.1.1006.3.2.1' => 'hostname:ACME-GV-023',
  '1.3.6.1.4.1.1006.3.2.2' => 'domain:pub.acme.com'
  '1.3.6.1.4.1.1006.3.2.3' => 'somevar:someval',
},
```

Mandatory options:

```
    -n <node>        hostname of the node
    -a <nms_addr>  send the trap to this address
```

Optional options:

```
    [-e <enterprise>] the enterprise OID to use (default 696)
    [-v <var_oid>]    the variable OID to use (default 1.0)
    [-m <message>]    the OID message text
    [-s <specific>]   the specific IOD to use (default 0)
```

signal_json

A RESTFUL JSON call can be added to the scenario using signal_json.

Mandatory options:

```
    -t <target> Define the target application (server and url)
```

Optional options:

```
    -p "variable=value"   Provide variable-value pairs to include in Json
post
    -n <node>             Provide the nodename
    -m <message>          Provide the message you wish to include
    -a <action>]          Provide the action value
    -s <status>           Provide the status
```

This call is dependent on the configuration file `/opt/yce/etc/`**`signal_json.conf`** and can be modified using the `edit configs` page under system. If the configuration file doesn't exists yet, it will be created the first time the `signal_json` command is executed from a scenario.

This configuration file must be setup to define the various `target` applications. These targets represent for example the ticketing system, the monitoring tool or the cmdb. Each of these targets have attributes like: host, URL path and authentication details.

since these details may vary per NetYCE server, these targets have their attributes defined per NetYCE hostname, or when they are identical to all servers, use the 'default'.

Example **signal_json.conf** file:

```
our $json_config  = {
    'ticketing' => {
        'default' => {
            host => 'http://1.2.3.4:8080',
            url => '/api/v1/services',
            # provide for websevice basic authentication
            username => undef,
            password => undef,
            # provide for webservice bearer authentication
            webtoken => 'abcdef',
        },
        'genesis' => {
            host => 'http://1.2.3.4:8888',
            url => '/api/v1/services',
            # provide for websevice basic authentication
            username => 'my_username',
            password => 'my_passwd',
            # provide for webservice bearer authentication
            webtoken => undef,
        },
    },
    'cmdb' => {
        'default' => {
            host => 'http://11.12.13.14:8080',
            url => '/api/v1/services',
            # provide for websevice basic authentication
            username => undef,
            password => undef,
            # provide for webservice bearer authentication
            webtoken => 'abcdef',
        },
    },
};
```

The JSON dataset has a few pre-defined attributes (node, message, status, action) but can easily be extended using the **-p** options. These -p options allow you to add any number of additional attributes.

To allow nesting of attributes (called hashes), the parameter name in the -p option may be using dots (**.**). These dots separate the key names used in the hash.

A simple example may clarify their use:

```
signal_json -t ticketing  -n <node> -m "going down" -p "moo.one='one
value'" -p "moo.two=two value" -p moo.three="three value" -p
"moo.four.one = 'four value=none"

Results in:
   using Json config for target 'ticketing' and server 'default'
   #--- JSON ----------------------------------------
   {
   "jobid" : "0619_0031",
   "target" : "ticketing",
   "action" : "",
   "status" : "",
   "node" : "ASD-CR02001",
   "message" : "going down",
   "moo" : {
      "three" : "three value",
      "one" : "one value",
      "two" : "two value",
      "four" : {
         "one" : "four value=none"
      }
   }
}
   Submit to webservice: http://1.2.3.4:8080/api/v1/services
```

st_exec

ST_EXEC allows the job to execute a **Service-type** from the scenario. Traditionally, changes are first prepared before scheduling a job. The preparation might include using Service-types to automate it. But in some cases the sequence of events must be reversed (e.g. deleting an object from the database before removing it from the network), or the choice and parameterization of the Service-type is dictated by the job/scenario status.

Any Service-type can be used with 'st_exec' provided the operator has sufficient permissions for the Client_type, or the call will be rejected at run-time.

If the Service-type uses API 'custom' variables, these MUST be included using '-p “var=value”' options or the call will be rejected at run-time. The logs, list the located custom vars of the Service-type, and any missing will be logged in the error message.

```
[-b <client_type>]      client_type of Service-type to execute. Overrides
client_type of node
[-c <service_class>]    service_class of Service-type to execute.
Overrides service_class of node
[-s <service_type>]     service_type of Service-type to execute.
Overrides service_type of node
 -t <service_task>      mandatory service_task of Service_type to execute

[-n <node>]             optional nodename (YCE nodes only) to provide the
-b, -c, -s options.
                         Also sets 'current' service, client and node
[-a <ipv4_subnet>]      optional ipv4 subnet/prefix in service-key or
global ipv4-subnet-id.
                         Sets 'current' subnet. Requires -n, excludes -6
[-6 <ipv6_subnet>]      optional ipv6 subnet/prefix in service-key or
global ipv6_subnet-id.
                         Sets 'current' subnet. Requires -n, excludes -a

[-p <parameter=value>]  define Service-type 'custom' parameters.
Mandatory for every custom used
```

```
                              in Service-type. Use multiple '-p'
```

The simplest call uses only the '-n node' and '-t service_task' options. By including the '-n node' the st_exec will use the client_type, service_class and service_type from the service the node is located in. If any of -b, -c or -s is specified, it will overrule the values found for the node.

```
[scenario]
Description <node> Add_vpn by ST

st_exec -n <node> -t "Add_vpn"
if <error>
    log -m "st_exec failed"
    stop
endif
```

Service-types using 'current'

When including the '-n node', some of the service-type commands that match on 'CURRENT' - representing the selected object in the GUI - will be set. The 'current' values for the service, client, site, and node are thus defined and allow the Service-type to be executed despite the lack of a GUI selection.

Note however, that 'current' subnet, ipv6-net or dhcp objects are not available by setting -n alone. Add the '-a ipv4net/prefix' or '-6 ipv6net/prefix' to 'select' the current subnet if the Service-type requires it. The subnet will be searched for using the service-key of the node. If the -a or -6 value is the ipv4/ipv6 subnet-id, then this subnet will be recognized as the 'current' subnet. The -a and -6 options can only be used with the -n.

API Service-types

A Service-type may also be fully specified by including the -b, -c -s, and -t options. This form is commonly used to execute API based Service-types. Since they usually require custom variables, these variables must be included using -p options. Before execution of the Service-type, all customs are located and logged. If -p variables are missing, the st_exec will not start and produces an error.

The example below shows the -p options one separate lines following the st_exec call. This is a supported format and is used to improve readability. Options are automatically joined if the first part of the line is a recognizable option ('-a ', '-6 '). Comments cannot be included.

```
[scenario]
Description Add_dmz_frw to LB

st_exec -b 'DC' -c 'LB' -s "api" -t "Add_dmz_frw"
 -p "frw_name = DC2-FW01"
 -p "lb_name = DC2-LB02"
 -p "pe_a_name = DC2-PE01"
 -p "pe_b_name = DC2-PE02"
 -p "vrf_id = 100"
 -p "vrf_template = dmz_vrf"

```

```
if <error>
    log -m "st_exec failed"
    stop
endif
```

It is allowed and sometimes useful to include a '-n node' option even when all four Service-task specifying options are explicitly set. In this case the node will provide the 'CURRENT' values for this node to be used in the Service-type. Of course, the use cases for this setup are very limited.

ansible_exec

This calls upon the local ansible installation. It allows the launching of either an existing Playbook, or the launching of a NetYCE Scenariobook by Ansible. Where the Playbook option refers to a pre-existing playbook-file that can be used as-is, the Scenariobook refers to the NetYCE template-based generated Playbook that can be parameterised using the NetYCE modelling and database.

Optional options:

```
    [-n <node>]           the node in question
    [-p <playbook.yml>]  Use pre-existing Playbook YML file - include
full path
    [-s <scenariobook>]  Use playbook created in a NetYCE job
    [-o "-vvvv"]         Use ansible options -
https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html"
```

Example, static yaml file:

```
ansible_exec -n <node> -p /opt/update_ntp.yml
```

Example, dynamic yaml file, which is generated by NetYCE:

```
config_create -n <node> -t update_ntp -f update_ntp.yml
ansible_exec -n <node> -s update_ntp.yml
```

script_exec

This calls upon a local script. Additional arguments can be provided to the script and if needed the interpreter can be overridden.

Optional options:

```
  -s script           Script to execute - full path may be included
  [-i <interpreter>]  Override interpreter to execute the script
  [-o <arguments>]    The script arguments. multiple are allowed
  [-t <timeout>]      The timeout in seconds before forcibly killing the
script. defaults to 300
```

Example, of the python script execution:

```
<output> := script_exec -i /usr/bin/python2.6 -s
/opt/scripts/some_script.py -o variable1 -o variable2
<output> := script_exec -s /opt/scripts/some_script.py -o variable1 -o
variable2 -t 120
```

List Operations

calc

Executes the given calculation

Mandatory options:

```
    -c <calculation> string containing the calculation, may be a variable
```

Example:

```
<valA> := 5
<newVal> := calc -c '5 + <valA>'

# Output in the job log:
Assignment: (valA -- result: (5))
Assignment: (newVal -- result: (10))
```

relation

Extract a variable (list) from a named relation, and returns this to a list variable. More information on relations.

Mandatory options:

```
    -n <node>      hostname of the node
    -r <relation>  name of the relation (context)
    -v <column>    the name of the relation column (variable) returned.
```

Optional options:

```
    [-p <parameter=value>]   provide additional relation variable=value
pairs
    [-f <column=value>   ]   define filter column=value pairs. The value
supports wildcards ('*' and '?')
                               the relation row must match all filters to
be included in the result
    [-l <limit>]             return the <limit> number of entries. The
limit must be > 0
```

Example:

Using variable: <Cpe_hostname>

```
    <CPE_Vrf_id> := relation -n <Cpe_hostname> -r Node_vrf -v Vrf_id -p
"Hostname=<Cpe_hostname>"
```

replace

Replace searches for a specific string-value and replaces it with another string-value. The replace value can be empty. The match string is non case-sensitive.

Mandatory options:

```
    -l <list> Provide the list value, it will use the first entry
    -c <match> The string that needs to match
```

Optional options:

```
    [-r <replace>] The match will be replace with this string
    [-a] Provide this to replace every match
```

Example:

```
<valA> := "abc"
<valB> := "def"
<newVal> := replace -l <valA> -c 'bc' -r <valB>

# Output in the job log:
Assignment: (valA -- result: (abc))
Assignment: (valB -- result: (def))
Assignment: (newVal -- result: (adef))
```

sort

Takes one or more lists and sorts its elements ascending. Returns a list variable.

Mandatory options:

```
    -l <list> a list variable
```

Optional options:

```
    [-r] reverse sort
    [-n] sort the list numerically
    [-u] return unique elements
```

Example:

```
    <listA> += "1"
    <listA> += "2"
    <listA> += "3"

    <listB> += "3"
    <listB> += "5"
    <listB> += "2"
    <listB> += "4"
    # sort both lists numerically in reverse keeping only unique items
    <sorted> := sort -n -u -r -l <listA> -l <listB>
    <msg> = concat -s ", " -l <sorted>
    log -m "items: <msg>"
    # prints the log message:
    # 2019-03-14 11:43:35 items: 5, 4, 3, 2, 1
```

concat

Takes one or more lists and concatenates its values in a single string. Each value is separated by the separator string specified with the '-s' option. Returns a list variable with a single string value.

Mandatory options:

```
    -s <separator>  separator string, default to a space (' ')
    -l <list>       a list variable
```

Example:

```
    <listA> += "1"
    <listA> += "2"
    <listA> += "3"

    <listB> += "one"
    <listB> += "two"
    <listB> += "three"
    <msg> = concat -s ", " -l <listA> -l <listB>
    log -m "items: <msg>"
    # prints the log message:
    # 2019-03-14 12:17:52 items: 1, 2, 3, one, two, three
```

like

> The 'like' command was removed in version 7.1.1. This command is superseded by the 'grep' command which supports both regex and wildcard formats for the condition.

grep

Takes one or more lists and a condition, and runs that condition against a wildcard or a regex over the list. Returns all matching elements as a list variable.

Mandatory options:

```
    -l <list>  a list variable
    -c <condition> a wildcard string: "some*ing" or a regex string: "/some.*ing/"
```

Other Options:

```
    [-u]       returns unique entries in the result. Duplicates are dropped, but only if the entries are really identical (case-sensitive).
```

NOTE: the difference for the regex. It requires the two '/' characters to mark the regex pattern (`/pattern/`). It will be always be case sensitive unless the 'i' modifier is used after the last slash: `/pattern/i`.

NOTE: Variables that were defined as [parameters] will be substituted with the '-c' argument. Run-time variables will not, but can be as a condition directly.

```
<filter> := grep -l <list> -c "*<var>*" # will only work when <var> is defined in the [parameter] section
<filter> := grep -l <list> -c <var>  # will work with run-time variable <var>
```

Example:

```
[parameters]

[scenario]

  Description <node> grep test

  # fill a list with relation entries
  # (must use += because each relation entry will copy this line in the final task before the scenario starts)
  <net_names> += <net_name@vlans>

  # copy this list to another list
  <all_names> := <net_names>

  # append some entries to this list
  <all_names> += "eenie"
  <all_names> += "meenie"
  <all_names> += "mynie"
  <all_names> += "moo"
  # filter the entries ending in "nie" using REGEX
  <nie_names> := grep -l <all_names> -c "/.*nie$/"
  # or using wildcard:
  # <nie_names> := grep -l <all_names> -c "*nie"

  # and log them one by one
  foreach <name> in <nie_names>
    log -m nie_name: <name>
  endeach

  end
```

The last lines of the log shows only ‘eenie’, ‘meenie’ and ‘mynie’ are grepped. The -c “nie” would have retrieved the same entries but might also have found ‘denied’.

match

Takes one or more lists and a condition, and runs that condition against a wildcard or a regex over the list. Returns a list with the matching substring(s) of each element.

The behavior of '**match**' is similar to the '**grep**' function but with an essential difference: where the 'grep' returns the actual element of the matching condition, the 'match' returns only the ***matching*** string. In essence, the 'match' is a grep function and a substring function rolled into one.

Mandatory options:

```
    -l <list>  a list variable
    -c <condition> a wildcard string: "some*ing" or a regex string: 
"/some.*ing/"
```

Other Options:

```
    [-u]       returns unique entries in the result. Duplicates are 
dropped, but only if the entries are really identical (case-sensitive).
```

As with 'grep', the 'match' supports both wildcard conditions and regex conditions.
The wildcard condition support the '*' and '?' characters to match multiple or single characters.
The wildcard condition uses a format like `-c “va?ue*”`. Wildcard conditions are case-insensitive.
The regex condition uses a format like `-c “/val.ue.*/i”`. The regular expression itself is positioned between slashes ('/…/') and optional modifiers are appended after the last slash. The 'i' modifier causes the matching to be case-insensitive.

Example:

This example illustrates the difference between the 'match' and the 'grep' functions. It uses the wildcard format for the condition. Grep only returns if there is a full match on the entire string. Match will get return the matched portion.

```
[scenario]
Description test match function

<list1> := "the first element"
<list1> += "the second element"

<list2> := "The third one"
<list2> += "The last one"

<grepped> := grep -l <list1> -l <list2> -c "*first*"

foreach <entry> in <grepped>
    log -m "grep entry: <entry>"
endeach

<matched> := match -l <list1> -l <list2> -c "*first"

foreach <entry> in <matched>
    log -m "match entry: <entry>"
endeach

end
```

Results:

```
The grep entries:
  "the first element"

The matched entries
  "The first"
```

When using the regex condition a distinct feature can be used: multiple substring matches. Regular expressions have the capability to 'mark' matched strings using '()' and refer to them by position ($1, $2 and so on). The 'match' function allows you to use this feature to use these marks in the condition regex and have these marked strings returned as a separate entry in the result.

When using these '()' markers, only those matches are returned. Without these markers the matching part of the condition is returned as was the case in the example above.

Example:

```
The same dataset as above using the regex condition below:
<matched> := match -l <list1> -l <list2> -c "/the (.*) (.*)/i"

Results in the matched entries:
  "first"
  "element"
  "second"
  "element"
  "third"
  "one"
  "last"
  "one"

And, when adding the optional '-u' creates a sorted list of unique
values:
<matched> := match -l <list1> -l <list2> -c "/the (.*) (.*)/i" -u

  "element"
  "first"
  "last"
  "one"
  "second"
  "third"
```

NOTE: Variables that were defined as [parameters] will be substituted with the '-c' argument. Run-time variables will not, but can be as a condition directly:

```
# this will only work when <var> is defined in the [parameter] section or
# is a parameter that is defined in de <node> context (standard template
variables).
<found> := match -l <list> -c "*<var>*"

# this will work with run-time variable <var> that was defined in the
scenario section
<var> = "value*"
<found> := match -l <list> -c <var>
```

NOTE: Regular expressions support the use of 'shorthand' character groups. A commonly used one is `\s` to define a whitespace (being space or tab). When using these in a condition, you have to protect these backslashes with a backslash:

```
Using spaces:
<found> := match -l <list1> -l <list2> -c "/the (.*) (.*)/i"
Using \s:
<found> := match -l <list1> -l <list2> -c "/the\\s(.*)\\s(.*)$/i"
```

split

Takes list(s) and a regex or wildcard separator to split the element strings into more elements; returns a list with the separated substring of each element.

Mandatory options:

```
    -l <list>       the list variable in question; multiple -l allowed
    -c <separator>  a wildcard separator string, 'var*ble' or a regex
string '/\svar(.*)\s/i'"
```

Other Options:

```
    [-n <number>]   limit number of returned elements
    [-u]            return unique elements as a sorted list
```

Example:

```
  <family> := "father, mother, son, daughter"
  <family> += "grandpa, grandma,uncle , aunt"

  # one way using regex
  <members> = split -l <family> -c "/\s*,\s*/"

  # another using wildcards
  <members> = split -l <family> -c ", "

  # same, but with a result limit of 1.
  <members> = split -l <family> -c ", " -n 1
```

Misc

pingable

Test if node is pingable

Mandatory options:

```
    -n <node>    hostname of the node
```

Optional options:

```
    [-a <address>]  override the node address
```

reachable

test if node is reachable using ssh or telnet

Mandatory options:

```
    -n <node>    hostname of the node
```

Optional options:

```
    [-a <address>]  override the node address
    [-p <port_list>] override the default ports "22, 23"
```

hangup

Disconnect node connection(s).

Mandatory options:

```
    -n <node>    hostname of the node
```

Optional options:

```
    [-c <connect>]  use 'console', 'api' or 'management' connection
```

wait

Delay all actions for <time> seconds.

Mandatory options:

```
    -t <time>   time in seconds
```

update_rev

Update database with active template revision.

Mandatory options:

```
    -n <node>    hostname of the node
```

Database

shortest_path

Locate the devices that make up the shortest path between end points. Returns a variable list of all nodes making up the path including the start and end points.

The shortest-path uses the topology (links between ports) of the nodes in the YCE database. Tracing a path is therefore limited to YCE nodes and the accuracy of the topology.

For finding the shortest path the Dijkstra greedy algorithm is used. The model uses any kind of topology between ports of different nodes as a valid path and takes into account the modelling of the hierarchy in that uplink directions have a cost of '1', downlink directions a cost of '3' and interconnections a cost of '2'. Thus resulting in a path where uplinks are preferred and core routes are not circumvented.

The algorithm will find exactly one path that has the lowest cost even is there are multiple paths due to redundancy.

Mandatory options:

```
    -s <node-name>   Starting YCE node name of path
    -e <node-name>   Ending YCE node name of path
```

Optional:

```
    -a <avoid-string>   String of ordered node-names defining links to
avoid using a higher cost
    -a [<avoid-list>]   Converts the <avoid-list> variable in a separated
string
```

The '-a' option is intended to find an alternate, secondary path between the start and end nodes once the primary shortest path was found. Using this option, the primary path is passed as an argument in a second call to 'shortest_path' where it is used to increase the cost of the links along this path. These higher costs will allow the algorithm to find an alternative path if it has a lower cost.

```
<path_pri> := shortest_path -s "switch1" -e "switch2"

<path_sec> := shortest_path -s "switch1" -e "switch2" -a [<path_pri>]
```

The example shows the use of the special notation “-a **[<path_pri>]**” that is needed here. Where the regular form (“-a **<path_pri>**”) to pass an option to a command takes only the FIRST element of the variable, this form concatenates ALL elements of the variable into one string. The use of '[..]' simplifies passing all elements separately using multiple '-a' options or having to convert the list into a string.

When using the '-a' option, the nodes names must form an ordered path. Also when multiple '-a' options are used, the resulting names are combined in a single path. Any '-a' value may be a single node name or a separated string of node names where the separator may be a space, a comma, a semicolon or a vertical bar. If a topology link exists between two neighbouring node names, the cost of that link will be increased by 5.

Note that the increase in cost is unidirectional: the cost of a path in the opposite direction remains unaffected and could therefore still be included in the resulting secondary path. In this and similar cases where a specific link or direction is to be avoided as well, the given path can be extended by including additional sets of nodes. Although not part of the primary path this will result in a higher cost for the link between these two nodes in the given direction. This manipulation of costs can also be used to find the primary path.

In the example below the primary and secondary paths must avoid the link between “switch1” and “switch2” in either direction:

```
<path_pri> := shortest_path -s "switch1" -e "switch2" -a "switch3
switch4" -a "switch4 switch3"

<path_sec> := shortest_path -s "switch1" -e "switch2" -a [<path_pri>] -a
"switch3 switch4" -a "switch4 switch3"
```

db_query

Retrieve a variable(-list) from the database. Returns a variable list.

Mandatory options:

```
    -t <table>   table to retrieve information from.
    -f <field>   the field to put in the return variable
    -w '<where>' where-arguments (for example: 'ClientCode="1000"'
```

db_update

Set a value in the database using a sql statement.

Mandatory options:

```
    -t <table>             table to retrieve information from.
    -p '<parameter=value>' parameters that need to be set
    -w '<where>'           where-arguments (for example:
'ClientCode="1000"'
```

> Make sure you put the entire part after -p and -w in quotes.

In case you wish to update Custom Attributes, the where would contain pipes ('|'). These would be interpreted as conditionals and need to be escaped using a '\'. Example:

```
db_update -t Par_vals -p "Var_value=<Var_value>" -w
'Par_key="<ClientCode>\|<SiteCode>\|<Hostname>" AND Var_name="var_name"'
```

Logging

For the logging, as well as the scenario in general and templates, it is possible to show/use the job_id using `<jobid>`.

log

Insert timestamped message in the log of the job.

Mandatory options:

```
    -m "<message>"    a string of text, in between quotes.
```

log_action

Add entry to action log.

Mandatory options:

```
    -n <node>            hostname of the node
```

Optional options:

```
    [-a <action>]          name the action type to list
    [-m "<message text>"]  the message to log. Make sure you use quotes
around this variable.
```

send_email

Send email notification to the operator/group mail addresses, if known.

This behaviour can be adjusted using the send_email_to tweak.

Optional options:

```
    [-s <subject>]          the email subject
    [-m <message_text>]     the email message text
    [-t <mail to>]          the email address to send email to.
    [-f <mail from>]        the email address to send email from.
    [-a <attachment-file>]  filename to include as csv attachment.
multiple -a are allowed
    [-r <custom-report>]    custom report to include as csv attachment.
multiple -r are allowed
```

Multiple '-t <email-recipient>' options are allowed, as well as a single -t option with a (quoted) list of email-addresses.

Example

See the entry below on `csv_file`

csv_file

Create csv file of list variable(s)

Mandatory options:

```
    -r <name>          the csv file name to be created in the job
directory. Spaces and special characters are replaced with '_', and
'.csv' is added
    -v <var_name>'     variable name with a list of values to form a
```

```
report column. add a '-v' for each column
```

Optional options:

```
    [-s <separator>]   separator character or string between columns.
defaults to comma (',')
    [-d]               change file format from unix (lf) to dos (crlf)
```

The generated csv report will be created in the job-directory named after the job-id (/var/opt/yce/jobs/<job_id>) and will receive the `.csv` extension. This file will be displayed in the pop-up window when viewing the job files. It includes a link to download the file.

Tip: The report file can be emailed directly from the job scenario to the operator by using the `send_email` command by attaching the report file to the email (the '-a <file>' option was added for this purpose).

Example

The example below uses data from the relation “vlans” to fill three list variables. The `csv_report` then creates a report from this data using these variables. Note that the report inserts the variable names as the csv-header using the casing as typed, in this case with a leading capital. Non-existing variables or empty variables included in the report generate no error but are included as a blank column.

The report name can include variables as well as is demonstrated. The -d option appends todays date to the report name.

The same report is then created as a local csv-file using `csv_file`. The '-d' option now converts the standard unix file into dos.

Finally the report is sent to the local 'yce' user using `send_email`. To demonstrate the send_email attachment file handling two more files are attached. Here, the '.csv' and '.log' extensions are tried to locate the report and log files in the job directory. But for other extensions the proper name must be used.

```
[scenario]
Description <node> mail test

<net_names> += <net_name@vlans>
<net_vlans> += <vlan_id@vlans>
<net_addrs> += <net_address@vlans>

csv_report -r "<node>_nets" -v Net_names -v Net_vlans -v Net_addrs -v
Nope -d

csv_file -r "<node>_nets" -v Net_names -v Net_vlans -v Net_addrs -v Nope
-d

send_email -a "<node>_nets" -a "<jobid>" -a "<jobid>.tsk" -t
yce@genesis.netyce.org
```

```
end
```

When executing the following logs were produced:

```
28-Function (csv_report -r "ASD-CR98001_nets" -v Net_names -v Net_vlans -
v Net_addrs -v Nope -d)
   saved report as 'ASD-CR98001_nets-20191217'
29-Function (csv_file -r "ASD-CR98001_nets" -v Net_names -v Net_vlans -v
Net_addrs -v Nope -d)
   saving report to '/var/opt/yce/jobs/1217_0053/ASD-CR98001_nets.csv'
30-Function (send_email -a "ASD-CR98001_nets" -a "1217_0053" -a
"1217_0053.tsk" -t yce@genesis.netyce.org)
   including 'ASD-CR98001_nets.csv' as attachment
   including '1217_0053.log' as attachment
   including '1217_0053.tsk' as attachment
   send_email: '' sent to 'yce@genesis.netyce.org'
```

csv_report

Create custom csv report of list variable(s)

Mandatory options:

```
    -r <name>          the report name to be added to the custom reports
    -v <var_name>'     variable name with a list of values to form a
report column. add a '-v' for each column
```

Optional options:

```
    [-s <separator>]   separator character or string between columns.
defaults to comma (',')
    [-d]               add the creation date to the report name
('_yyyymmdd')
```

The generated csv report will be added to the `Operate - Reports - View reports` menu and is subject to the same cleanup and ageing process as the other reports. The report can be retrieved using the 'fetch_report' API call and/or the download URL.

Example

See the section above on `csv_file`

Error handling

All scenario commands can set the 'error' flag if their execution is considered failed. When this flag is set (an error occurred) it will normally have no effect. Only when the scenario checks for it can it affect the further execution of the scenario. To stop the scenario execution and mark it 'ABORTED' the **`stop`** command MUST be included.

When the scenario reaches the **end** command (that is always present at the bottom of the scenario) the job status will be 'SUCCESSFUL' regardless of any errors it encountered. Therefore a scenario that contains no error handling or a `stop` command will by definition be successful.

```
reachable -n <node>
if <error>
   log -m "node <node> is non-responding"
   stop
endif

cmd_exec -n <node> -f <node>.cmd -v
if <error>
   log -m "command failure"
   stop
endif

end
```

Since having to include explicit error checking for every command in a lengthy scenario can be quite tedious, the default error behaviour can be modified to automatically ABORT the scenario at the failing command. This is accomplished with the **`stop on-error`** command. The above could then be simplified as:

```
stop on-error

reachable -n <node>

cmd_exec -n <node> -f <node>.cmd -v

end
```

Note that the automatic abort will ignore any error handling that is present in the scenario since the scenario will stop before it reaches the next line with the 'if error' test.

At the point where the normal error handling by the scenario must be resumed the **`stop default`** command can be inserted. This allows for explicit error handling where it is needed and skip it where it might be implicit.

```
stop on-error

reachable -n <node>

stop default

cmd_exec -n <node> -f <node>.cmd -v
if <error>
   log -m "command failure"
   stop
endif
```

```
end
```

Here documents

At times it is cumbersome having to create and define lots of trivial or very small templates. In those cases the scenario command can support HERE DOCUMENTS. This name is a reference to unix systems where multi-line input can be given for a command. In our case that input represents the content of a template.

A here document uses the **< <** and a terminator string that is defined at the end of the command line. The next line all the way up to the terminator string on a line by itself then forms the here document that will be used as the template text.

An example using the scenario command **`cmd_exec`**.
The string EOT is used here as terminator (end-of-text), but any string is allowed.

```
description <node> here document test

cmd_exec -n <node> -f here.cmd <<EOT
!
  a sample command for hostname <node> of customer <ClientCode>
  another command for the node in <SiteCode>
  the full template syntax can be used in a here-document
  | condition | {template}
!
EOT
```

This example will create the command file 'here.cmd' with the content below and then will execute these lines one-by-one on the node's cli:

```
!
a sample command for hostname AMS-DC99001 of customer ACME
another command for the node in AMS-DC99
the full template syntax can be used in a here-document
!
```

Please note that in the example above the `cmd_exec` was invoked with the option `-f here.cmd`. An explicitly named output file is essential here since the generated cli commands from the here document will be stored in this file.

The second example shows how a more complex command file can be created using a few `config_create` commands, with and without the use of here documents. The idea is that the command file is created bit-by-bit over the execution the scenario using command or config parsing.

Since the default behaviour of the `config_create` command is to restart the config file on each invocation, the `-x` ('extend') option must be included to append new configuration lines.

```
description <node> here document test

```

```
config_create -n <node> -f my.cnf <<EOT
#
  SOME COMMANDS FOR <node>
#
EOT

config_create -n <node> -f my.cnf -x -t clear_atm

config_create -n <node> -f my.cnf -x <<EOT
#
  SOME MORE COMMANDS FOR <node>
  {create_atm}
#
EOT

cmd_exec -n <node> -f my.cnf
```

In this case the finally resulting config file 'my.cnf' is then executed on the node.

Deprecated

These commands are still supported for backwards compatibility, but they are not meant to be used in new scenarios. When upgrading to version 7.0.2, a conversion was executed whereby the old commands and options were replaced with the new. Creating scenarios using the old commands is still supported for the time being.

import_cfg

Execute command line-by line on cli of a NetYCE managed node.

Mandatory options:

```
    -n node      hostname of the node
    -f cmd_file  file containing the commands you want to execute
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]  overrides the management domain of the node
```

basic_import

Execute command line by line on cli of a node in the cmdb table - does not support templates.

Mandatory options:

```
    -n node      hostname of the node
    -f cmd_file  file containing the commands you want to execute
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]  overrides the management domain of the node
```

gen_startup

Generate commands from template.

Mandatory options:

```
    -n node      hostname of the node
```

Optional options:

```
    [-t <template>]              overrides the default main-template of
the node
    [-p <parameter=value list>]  define or overwrite parameters for use
in template
    [-i <interface-list>]        generate the config(s) for the portnames
listed
```

write_startup

Upload full config file and make startup.

Mandatory options:

```
    -n node      hostname of the node
    -f cfg_file  file containing the config you want to upload
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
```

save_config

Save config on device and download to nccm.

Mandatory options:

```
    -n node      hostname of the node
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
    [-m <message>]  adds a message to the nccm as the cause
```

diff_config

Fetch running config and compare to latest nccm config.

Mandatory options:

```
    -n node      hostname of the node
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
```

reload

Restart the device.

Mandatory options:

```
    -n node      hostname of the node
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
```

restore

Upload config from nccm and reload.

Mandatory options:

```
    -n node      hostname of the node
```

Optional options:

```
    [-a <address>]  overrides the address of the node
    [-d <domain>]   overrides the management domain of the node
```

LogAction

Add entry to action log.

Mandatory options:

```
    -n <node>    hostname of the node
```

Optional options:

```
    [-a <action>]  name the action type to list
    [-m <message text>]  the message to log
```

trap

Send snmp trap from yce server to nms (“1.3.6.1.4.1.696.1.0 - 0”).

Mandatory options:

```
    -e <enterprise> the enterprise OID to use (default 696)
    -v <var_oid>    the variable OID to use (default 1.0)
    -s <specific>   the specific IOD to use (default 0)
```

Optional options:

```
    [-m <message text>] the trap message
```