Table of Contents

Template edit

The templates form can only be opened when a client is selected in the Main Build operations section. All the templates from that client type will be loaded.

The syntax for the templates is not checked for any errors. NetYCE is not an expert system. The templates are made of the actual configurations for the nodes in a parametrized way. This means, the configuration should be created by engineers who have intimate knowledge of the used devices and their configurations.

Variables within a template

Variables are surrounded by '<' and '>', thus <variable>.

Comments within a template

It is possible to use comments within a template. This is done by using one of the following characters for each line of comment:

- **#**
- **!**
- **--**
- **//**

Dependant on the text delimiter of the vendor (module), the line will be ignored or generated by the configuration generator. Variables will be substituted and checked in a comment rule.

Example

The following template:

```
# Specific SNMP server configuration
snmp-server contact <Snmp_syscontact>
! SNMP server 1
snmp-server <Snmp_traphost1> <Snmp_trap_community>
! SNMP server 2
snmp-server <Snmp_traphost2> <Snmp_trap_community>
```

For vendor **Cisco_ios** it will result in:

```
snmp-server contact info@netyce.com
! SNMP server 1
snmp-server 192.168.1.42 public
! SNMP server 2
snmp-server 192.168.33.89 public
```

For vendor **Huawei_S** it will result in:

```
# Specific SNMP server configuration
snmp-server contact info@netyce.com
snmp-server 192.168.1.42 public
snmp-server 192.168.33.89 public
```

Using a subtemplate within a template

Subtemplates are used within a template by using the template name between curly brackets { } like the example:

```
# Next the first subtemplate will be loaded.
{SubTemplateName1}
# Now the second subtemplate will be loaded.
{SubTemplateName2}
```

This will be the actual output when the template will be generated:

```
These are the contents of the first subtemplate
!
These are the contents of the second subtemplate
!
```

Subtemplates can also be used within a subtemplate.

Main templates

Main templates are mostly used to assemble all the needed Sub templates. Main templates should not contain any actual configuration. All main templates are linked to a specific Node class. Node classes are defined with the General settings by defining a new Node type. Also a Main template must have a hardware vendor and type.

Sub templates

Subtemplates can be used to group partial configurations, such as ACL's, SNMP etcetera.

Subtemplates can also be used from within a Subtemplate. This will result in the following example:

mainTemplate.txt

```
...
{managementSubtemp}
...
```

The management subtemplate contains the following:

managementSubtemp.txt

```
#Subtemplate for all management related configuration
{snmpSub}
{sshSub}
{syslogSubt}
```

And the subtemplates snmpSub, sshSub and syslogSub will contain the actual configuration. Nested subtemplates are limited.

Automation templates

Automation templates are available from the *Sub templates* tab. Automation templates are sorted below the Sub templates.

Port templates

Port templates are used to configure ports. A port type must be entered and port templates can only be assigned to ports to which the hardware type is a match. (i.e. A FastEthernet port can only be assigned a FastEthernet port template)

Parsing templates

Parsing templates are used in conjunction with the parse_cmd scenario command to extract data from a node. The parsing template syntax allows for several ways of extracting data.

Relations

Main article: Relations and named context

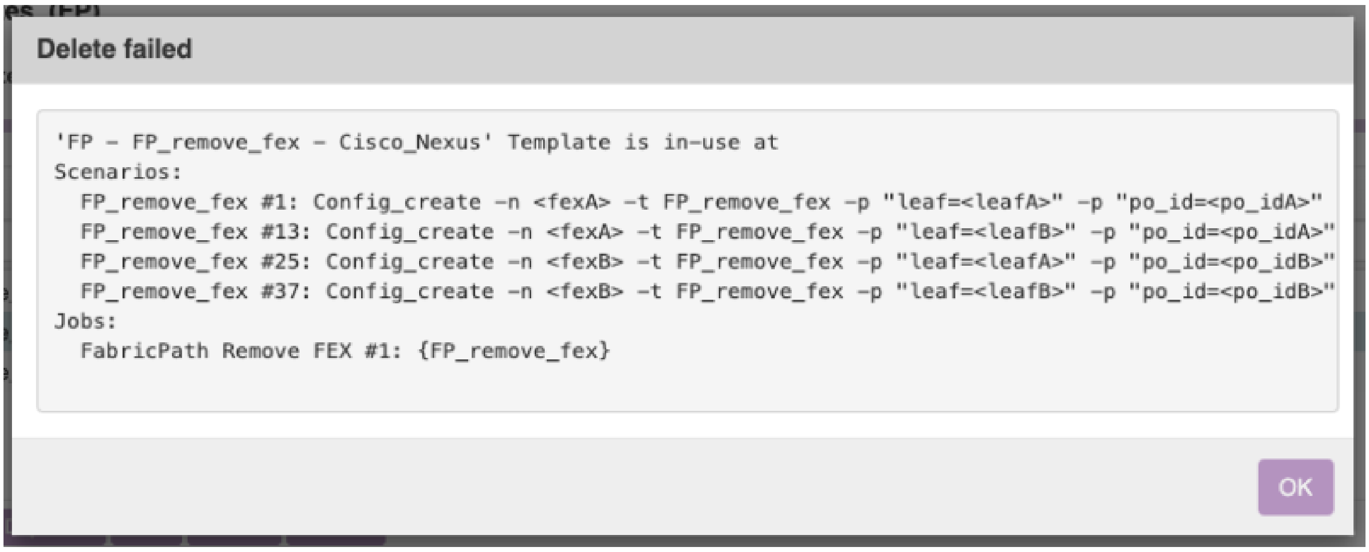
All the nodes within netYCE are created from a hierarchical model of objects. By using the relation between these objects, very short templates can be created with a lot of information in them. This is done with relations.

Dependencies

Templates can be used in other Templates, Scenarios and Stored jobs. To prevent deleting a Template that is in use in any of these applications, a dependency check is performed when attempting to delete a Template.

The results of the dependency check is a detailed report on which templates, scenarios and jobs include a reference to the Template and on which line number. The line with the reference is incorporated.

For Main-templates and Port-templates, the report will also include any nodes using the template for their configuration. In the case of Port-templates, an additional check on the Main-template port blueprints is included. Any Main-template using the Port-template to be deleted in their “Switch ports” blueprint will be reported.



```
'FP - FP_remove_fex - Cisco_Nexus' Template is in-use at
Scenarios:
  FP_remove_fex #1: Config_create -n <fexA> -t FP_remove_fex -p "leaf=<leafA>" -p "po_id=<po_idA>"
  FP_remove_fex #13: Config_create -n <fexA> -t FP_remove_fex -p "leaf=<leafB>" -p "po_id=<po_idA>"
  FP_remove_fex #25: Config_create -n <fexB> -t FP_remove_fex -p "leaf=<leafA>" -p "po_id=<po_idB>"
  FP_remove_fex #37: Config_create -n <fexB> -t FP_remove_fex -p "leaf=<leafB>" -p "po_id=<po_idB>"
Jobs:
  FabricPath Remove FEX #1: {FP_remove_fex}
```

Depending on user permission level, the dependencies that prevent deletion can be overridden. If permitted, the dependencies report concluded with the message “Do you want to delete anyway?” and offers the choices 'Cancel' and 'Ok'.

The permissions are defined in the YCE.Auth_permissions table and can be changed using the 'delete_override' actions for 'relations', 'templates', 'template_revisions' and 'scenarios'. The default is that only 'Manager' and 'System' level users have the override permissions (Role_5 and Role_6 columns).

Directives

Port-templates can fully use any Relation to retrieve (lists of) variables. However, some Relationships become port-specific by referencing <port_name>, <port_type> or any of the port-specific variables in the SQL of the Relationship. In these cases it is **absolutely necessary** to force the YCE configuration generator to re-query the Relationship for every port. Failing to do so causes the last results to be reused - probably those of another port.

This behaviour is subject to YCE template directives, of which the `#reload` is the most used. YCE directives must be the first word of the line and there may no whitespace between the hash (#) and the directive itself. Most directives require one or more arguments. Directives cannot be used on the same line as cli commands. Their arguments cannot be variables.

The **`#reload`** directive forces a Relation to be re-queried. Any previous results form the Relation are deleted. The #reload directive must be present BEFORE retrieving the variables of that relationship in the (port-) template.

The #reload directive attempts to load all arguments as Relations:

```
#reload port_connections vlans_attached
```

is equivalent to

```
#reload port_connections
#reload vlans_attached
```

The **`#load`** directive is similar to the #reload but does not delete the previous results.

The **`#use`** directive also queries a Relationship but immediately makes it the current context. The relation name can be omitted for the duration of the #use. It takes only one Relation as argument.

Lastly, the **`#wait`** directive although it has nothing to do with Relationships. It is used to create a momentary pause when executing commands on the command-line of a device. The sole argument to the #wait directive is the number of seconds to pause. This number must be a positive real number (e.g. 1 or 2.5, or 1.25).

When you execute a command which will take a while to process you'll want to wait a little while before executing the next command (otherwise you risk the second command not being executed at all), this can be accomplished using the directive “#wait n.n” which will wait n.n seconds on the CLI before the next command will be executed, make sure to use this directive **after** executing the command which will take a while to process.

The syntax is: “*#wait n.n*” everything after that will be ignored, spaces in front of the directive are allowed:

```
#wait 1.5 - will wait 1.5 seconds.

  #wait 1.5 - will also wait 1.5 seconds.

# wait 1.5 - will be seen as a comment.

#wait 1,5  - will be interpreted as a "#wait 1" because of the comma has
been used instead of a point.

#wait          - will also be seen as a comment.
#wait a second - will also be seen as a comment.

#wait 10 let's wait for a while  - will wait for 10 seconds.
```

There is no restriction on a maximum period, “*#wait 3600*” can and will wait for an hour. “*#wait 0.01*” also works. By default a waiting time of 0.2 seconds will be used between every command, issuing a “*#wait 0.1*” will result in a waiting time of 0.3 seconds until the next command will be executed.

When a full configuration for a node gets compiled the #wait will be omitted (download file), only in case of partial templates or commands the #wait will be passed on to the job.

Ports

Main templates posses ports. When creating a new node, these ports will be copied to the new nodes. To manage a main template's ports, click on the “Switch Ports”-button in the main templates tab.

In the ports subform, ports can be added (**+**), removed (Delete) and changed ( ). By clicking a single port, the port will also be edited. By looping a lasso around multiple ports, all the selected ports can be edited.

You can also add, edit and delete port slots. For now this only is used in order to specify the slot's port layout.

The port layout syntax is as follows: the number of rows, followed by an H(orizontal) or V(Vertical). If the port count should start at bottom-left (instead of top-left), include a U. If the port count should start at the top-right (instead of top-left), include a B, then a dash (-), followed by the number of ports that should be displayed per block. Example: 2V-12.