Relations and Contexts

Contexts

Default Context

When generating the configuration for a networking device, a large number of parameters are retrieved from the NetYCE database that are related to that node. These parameters are used in the Templates that make up the configuration and are substituted during generation. For each node type, the set of available parameters may vary. This set of parameters and their values we call the **Context** of the node.

The **Default Context** is the standard context of the node for which the configuration is generated.

Named Context

Additional parameter sets reflecting specific relations in the network design can be defined by the user with *Modeling* permissions. In order to create these relations between nodes and gain access to the corresponding parameters, *named contexts* are defined. These relations are being stored as an SQL query for which the corresponding parameters are retrieved.

A *Named Context* is the name by which the SQL query is being stored and is also the name of the context. A named context is then being accessed as a temporary table that, in combination with a parameter name results in one or multiple values.

The various named contexts become an extension of the default context from which the parameter values can be accessed via the name of the context. A named context is defined only once as a '*Relation*' by an YCE modeler in the form of an SQL statement and is accessible for all templates.

A configured named context or 'relation' is available for all templates, sub templates and automations. There are no restrictions with regards to the number of named contexts that can be configured or that are used in a template.

Current Context

The *Current Context* is always the Default Context of the node, but can be changed (temporary) under the influence of the Template-directives to a *Named Context*.

Parameters in the current context are expressed in the following form: `<param>`. This will substitute `<param>` in the template with the corresponding value. The substitution is executed exactly once even if multiple values exist. In this case the first value of `<param>` is used.

In case that all values of `<param>` need to be substituted, `<param@>` can be used. This implicitly refers to the current context. When substituting multiple values for a parameter exactly one line for each parameter value will be generated.

The default context is in fact also a named context with the name of the node. For this reason the creation of a named context with the name of the node is not recommended. The results will be confusing.

Context Directives

Context directives (or pointers) are used in the template text and can influence the context behavior while generating a configuration.

Load or reload a context

`#load context`
This forces the loading of a *Named Context*. Normally a named context is retrieved from the database at the moment of its first reference. The `#load` directive can force this occurrence. By doing so, the dependencies between named contexts can be forestalled, as the SQL statements of a named context can have parameters of another named context.

The context information is loaded in addition

`#reload context`
The `#reload` directive will be used primarily in port-templates. Contexts that are used in specific interfaces, will usually refer to <`port_name`> in their SQL definition. As these will change for each port, a `#reload` of this context in the port-template is a requirement. If this would've been done using a `#load` each <`port_name`> would be added to the previous set of information.

The context information is clean (previous information is cleared)

The `#load` and `#reload` directive can have multiple contexts as argument. These are then reloaded in the order from left to right.

Use a named context

`#use context`
`#use`
The `#use` directive appoints a named context as the default context. This can be useful to be able to use a sub-template for a different context than the current default was intended.

The use of these will be very sporadic since sub-templates with variable contexts are normally invoked via the `{subtemplate, context}` syntax that automatically performs this context exchange.

Without argument, the `#use` directive will restore the default context of the node.

Wait directive

`#wait <float>`

A wait directive also exists to allow for additional waiting between commands. This is explained in more detail at Templates.

Remarks

Context names, parameters and directives are not case sensitive.

De directives like `#load` and `#use` cannot be entered as `# load` or `# use`. Depending on the vendor type, the developer can enter comment lines in the Templates using `# ` as a leader string. Consequently using `# load some_relation` in Template will be evaluated as a comment, not a directive.

Named Context definition

A named context is being stored under the 'Relations' form of the YCE client front-end as an SQL query. This form is accessible via the Relations form:

Named contexts will most always contain a `WHERE` clause with a reference to `<Hostname>`, `<SiteCode>`, `<ClientCode>` or `<Domain>` as this forms the relation with the context of a node.

Transformed Context

When creating a named context (relation) the SQL definition of the relation can be modified to automatically create additional columns based on the values found in the SQL results.

This extension of columns is the result of transforming a field value from a selected column to a column name. The value(s) in that column then points to another selected value from the record. In that way, a result table in which every line describes a parameter, can be changed to a table in which this parameter-name becomes a field-name. That field name then receives one of the field values that belong to that parameter.

To extend the existing context to a Transformed context, a 'Transform' line is being added, preceding the context SQL query. This Transform line has the syntax:

```
TRANSFORM parameter-veld WITH value-field
```

This transform-line is directly followed by the `SELECT` query of the context.

A transformed context makes it easier to use parameter names directly in a template. Example: The reference to `<Server_address@Servers_client:'Dhcp_server'>` will return the parameter (colum name) `Server_address` of the relation `Servers_client`. Since the qualifier `Dhcp_server` is included, only the result rows where one of its values matches the string `Dhcp_server` will be included. Effectively substituting the server_address of the server named Dhcp_server.

Should a Transform be applied (using `Transform Server_name With Server_address`), the same result can be retrieved with
`<Dhcp_server@Servers_client>`

The transformed `Server_name` column created new columns named after the server-names, each with a value matching the `server_address`. The new column `Dhcp_server` has now has the value of the `Server_address`. The substitution will result in any server-address of the Dhcp server.

As the context is only extended with a number of new columns, both reference forms remain available (and can also be combined)

It is not possible to transform more than one parameter.

Relation Test

As of version 5.0, it is possible to develop a relation more interactive. The web-tool 'Relation Test' allows the developer to evaluate the outcome of a query for a specific node. The transform syntax is here being supported. It can be found in the Build → Relations menu section.

The Relation Test starts with selecting a node to examine the context results for a relation. By selecting various nodes the results for a relation for each of these nodes can quickly be examined.

The relations can be retrieved with the drop down 'Relations'. After activation with the 'View Context' button is the SQL query being shown. Also is the query being processed for the selected node and shown as a table.

Changes to the query can be added by editing the 'Context Query' field. By applying the 'Evaluation' button, this is being executed. In this way a context-query (or relation) can be changed iterative.

It's not possible to save a changed query directly. To do this, the final query needs to be pasted back in the Relations form of the YCE client.

Parameters

Current Context

Parameter substitution for the Current Context has two different forms

`<param>`
This substitutes the value <`param`> in the template line at the location mentioned. The context where this parameter refers to is the Current Context, that without applying `#use` almost always is the default context of the node. <`Hostname`> then refers to the name of the node for which the configuration is being generated.

`<param@>`
This generates a configuration line for all values (also non-unique) for the parameter from the Current Context. See also description of <`param@context`> mentioned below.

A reference to <`param@`> not only results in a configuration line for all values of param, but also forces <`param`> to be searched in the current context. In most cases this will be the default context of the node, but with the '#use context' directive, an arbitrary context can be made.

Named Context

The substitution of named contexts is different than for normal parameters.

A query of a named context often results in multiple records. Mostly, the relation between nodes consists of multiple elements such as switches or routers on different locations, ports of a switch, SLA measurements QoS parameters etc. In these cases a configuration line is preferred for all possible values of the parameter in this context.

`<param@context>`
This substitutes the value of 'param' of the mentioned context (relation-name) in as many configuration lines as there are records that have been retrieved with the named context. A template line where multiple parameters from the same context are used, results in configuration lines applying one record every time of the named context so the underlying relation is maintained.

Example:

Named Context: routers

```
Hostname    Lan_address    Lan_mask
switch1 100.216.80.23    255.255.254.0
switch2 100.21.100.134    255.255.254.0
switch3 100.21.120.3    255.255.254.0
```

The template line:

```
Remark <Hostname@routers> on adres <Lan_address@routers>
```

Results in:

```
Remark switch1 on adres 100.216.80.23
Remark switch2 on adres 100.21.100.134
Remark switch3 on adres 100.21.120.3
```

If multiple named contexts are used in one template line, then configuration lines are generated for ***all permutations*** of both contexts. In case a parameter is added in the above example for another named context, which resulted in two records, a total of six configuration lines are generated.

`<param@context:'key-value'>`
The named context parameter with a 'key-value' uses a given value (the 'key-value') as a filter. Only those relation records where ***any of its parameters*** matches the key-value will be included. Relation records that do not contain the value are ignored. The use of a key-value is not case sensitive. In case multiple records of the named query apply, then again multiple configuration lines are generated.

`<param@context:'*key-value'>`
`<param@context:'*key?value*'>`
The wildcard characters * and ? are allowed in with a key-value to be able to make non-exact

matches. Again, the match is not case sensitive.

Combining a key-value parameter and a normal parameter from the same context will result in the same behaviour as if the contexts were different: different configuration lines are generated for all permutations in both named contexts.

<param@context:param_y='key_value'>
<param@context:param_y='*key?value*'>
This format allows you to specify which of the parameters in the relation must match the key-value. Instead of trying to find the key-value in any of its columns, only the named one is tried. As with the other context based substitutions, the number of resulting lines will depend on the possible permutations. However, when the same 'context - param_y - key-value' filters are used within one line, the relation records will be treated as one set and will not result in permutations.

```
# safe to use without permutations:

vlan-name: <net_name@vlans> vlan-id: <vlan_id@vlans>

vlan-name: <net_name@vlans:client> vlan-id: <vlan_id@vlans:client>

vlan-name: <net_name@vlans:vlan_id=client> vlan-id:
<vlan_id@vlans:vlan_id=client>
```

Indirect

Parameters can also be substituted indirectly. This option is useful in case a reference in the form of a node name or template name is needed.

For normal substitution, the value of the mentioned parameter will be replaced by the string `<parameter>`. However, in case this value also has the form of a parameter reference (`<param2>`), then the reference will be substituted, provided a final value can be determined. We call this indirect substitution.

Indirect substitution is also supported for context parameters. Meaning that the referenced parameter points to a named context (`<param@context>`) and therefore can generate multiple lines.

The implementation however is not fully recursive. Only the first and second references can name a context, any subsequent indirect references must name regular parameters.

Template line

In all above mentioned paragraphs mentioned template line it is also possible to use next to direct commando’s the following:

```
{sub-template}
{template_ref,context}
{template_ref,context,conditional}
{template_ref,context,conditional,conditional}
```

Multiple conditionals can be added, which will make a logical 'AND'.

Commands with substitution of:

```
<param>
<param@>
<param@context>
<param@context:key>
[Eval(..)]
[IpAdd(..)]
[InvMask(..)]
```

In case of context references (parameters with a @), all references of the context will result in a line if a condition forces the template line to be used.

Line concatenation

For situations where a context results in multiple lines, it is sometimes possible that this is executed as a block instead of line by line. The can easily be realized using the concatenation sign \.

An example of a list of vlan interfaces with some details:

```
|<net_address@vlans> != ''| interface vlan <vlan_id@vlans>
 description <net_name@vlans> vlan
 network address <net_address@vlans>
!
```

This results in a useless configuration. Also the condition is limited to the first line so it should be repeated for every line.

```
interface vlan 100
interface vlan 150
interface vlan 200
description Inrol vlan
description OSPF_Vlan vlan
description Servers_bk vlan
description Native vlan
network address 10.106.63.204
network address 10.106.63.140
network address 10.106.47.32
network address
!
```

By using concatenation, this one does result in the desired outcome. The condition is now also applicable for the whole block so the Native subnet (without ip- address) is suppressed:

```
|<net_address@vlans> != ''| interface vlan <vlan_id@vlans> \
  description <net_name@vlans> vlan \
  network address <net_address@vlans> \
!
```

Results in the desired list:

```
interface vlan 100
  description Inrol vlan
  network address  10.106.63.204
!
interface vlan 150
  description OSPF_Vlan vlan
  network address  10.106.63.140
!
interface vlan 200
  description Servers_bk vlan
  network address  10.106.47.32
!
```

Sub-Templates

Sub-templates can be invoked from within the main template. This can be done iterative by nesting sub-templates. Also conditional use is possible.

Template substitution

```
{sub-template}
|<param>| {sub-template}
{<template_ref>}
```

Sub-templates are invoked and substituted by its name. The texts of the sub-template is being evaluated line by line, like the main-template and included in the configuration. A sub-template always uses the Current context.

Also a template can be invoked indirectly by using a parameter name in which the name of the sub-template is mentioned. Template names are not case sensitive. The template names or its reference may involve all template sorts: main, sub, automation, port.

Context-templates

In situations where sub-templates must be used several times, and by which each time a different set of values must be used, Context-template forms can be used.

```

{template, context}
{<template-ref>, context}
```

Here the template `template` is substituted whereby every time a subsequent record out `context` is made the current-context.

As an example it is easy to generate all Vlan interfaces with its own ip-addresses with e.g. `{Vlan interface, Vlans}`.

The `<template-ref>` may also be a context parameter from the same context as for which the sub-template is generated.

This is useful if the template name is mentioned in its context. Example:

```
{<Sla_template@Sla_monitors>, Sla_monitors}
```

This results in the execution of all defined Sla_monitors whereby the Sla_template parameter contains the template name of the context.

```
{<template-ref>, context, parameter=’value’}
```

This is the most complex use of sub-templates being supported. Here a condition is used for the context for which (indirectly) a template is being called upon. This is useful to generate the configuration of a specific port or set of ports.

```
{<Port_template@Port_map>, Port_map, Ifname=’FastEthenet0/0/0’}
```

This conditional use of context templates allow for conditions of the type `variable = value`. The “variable” must be an existing column in the relation used. The “value” given may contain wildcards ('*' and '?'). If the condition is to be used to exclude a (set of) relation records, then include an exclamation sign (`!`) with the = operator.

```
# include all GigabitEthernet interface variants (GB, 10G, 40G, 100G)

{ <port_template@interfaces>, interfaces, port_type = '*Gigabit*' }

# do loopbacks only

{ <port_template@interfaces_all>, interfaces_all, port_class = Lo }

# exclude ten-gigabit interfaces

{ <port_template@interfaces>, interfaces, port_type != ten* }

# Filter on dhcp_type and Net_address

{DHCP_pool, ipv4_customs_node, dhcp_type=2, Net_address=<Net_address>}
```

Configuration Generator

Version 2.0

The templates are being transformed into a specific vendor configuration or commands by the configuration generator. The generator is a Perl script that accepts the syntaxes as described in this document. The version 2.0 generator is optimized to minimize the load on the YCE database. Syntax from version 1.0 that are no longer supported, will be automatically converted to version 2.0 before send to the generator, but the templates itself are unchanged. Certain specials were not possible to convert and need to be changed in the template.

Command line

De generator can be found on the YCE servers in the directory `/opt/yce/bin` and is named `c2.pl`

YCE configuration generator version 5. Creates template-based full node configurations or command sets.

Usage: c2.pl

```
   -n, --node nodename
                  Mandatory argument.
                  Nodename must exist in YCE database
                  Any orphaned argument is taken as nodename
   -t, --template template[,template]
                  Defaults to node main template
                  Multiple templates are appended in sequence
                  Template '--' is read from STDIN
   -j, --jobid jobID
                  Optional job directory in /var/opt/yce/jobs/<jobID>
                  When provided, the log is created here,
                  otherwise logs to /var/opt/yce/logs/yce.log
   -v, --verbose [level 0..3]
   -h, --help

Output files:
   with jobID    - in directory /var/opt/yce/jobs/<jobID>
   without       - in directory /var/opt/yce/configs
   with template - filename is <nodename>.cmd
   without       - filename is <nodename>.cfg

Examples:
   Full node configuration in /var/opt/yce/0719_005/S1005011.cfg
   --jobid 0719_005 --node S1005011

   Dialer configuration for M1005011 in /var/opt/yce/configs/M1005011.cmd
   --template cn_dialer M1005011
   Dialer remove *and* add configurations for M1005011 in
/var/opt/yce/0719_005/M1005011.cmd
   -j 0719_005 -n M1005011 -t cn_dialer_del,cn_dialer -v

   Create configuration for FastEthernet port of S1005011 in
/var/opt/yce/configs/S1005011.cmd
   echo '{<Port_template@Port_map>, Port_map, Ifname=FastEthernet0/0/0}'|
c2.pl -n S1005011 -t --
```