Service-types definition

Service-type commands

It should be clear now that Service-types are an automated way to create and manipulate network related objects in an vendor independent (agnostic) and abstract manner. They do so by executing step-by-step a series of Service-type commands that perform specific actions on these objects.

Only four basic types of commands are needed: ADD, LOCATE, ASSIGN and DELETE. These execution types are available on all 17 object types and come in many specific variations to perform an action. In total about 250 unique Service-type commands are available.

Service-types commands are created using a simple wizard where the operator first chooses an execution type and an object type, which then filters for the available actions and options. Each Service-type command consists of of no more than six components.

The command for creating a new node in a service container looks like this:

```
[exec - class - scope - match     - value - alias]
 ADD  - NODE  - <srv> - NODE_TYPE - value - <node>
```

This line could be read as: ADD a NODE to the service <srv> with the NODE_TYPE ‘value’ and return the alias <node>

A subsequent Service-type command could rename the auto-generated nodename to something specific using:

```
[exec   - class - scope  - match     - value - alias]
 ASSIGN - NODE  - <node> - NODE_NAME - value
```

This line could be read like: ASSIGN to the NODE <node> the NODE_NAME ‘value’.

In this context, any enclosed in <..> refers to an ‘alias’. The alias is a descriptive handle to a returned object. It is used to perform additional manipulations and assignments once it is created or located. The notation using <..> is optional but highly recommended.

See the section below on Alias names Service-types overview for some best-practices recommendations.

> A full list of all available Service-type commands per object class is maintained in the article Service-type syntax

Service-type Objects

There are 17 object classes available for manipulation using Service-types in NetYCE. Each represents a building block of the ‘abstract network’. Some have an hierarchical dependency, others are stand-alone or are associated with another object.

Client
The Client object is a representation of a customer, internal or external that uses a specific Client-type (the architecture: site-types, Service-types, IP-plans, node-types, templates, stored jobs, …). The Client object has few attributes and is part of a Region and is itself assigned IP supernets (v4/v6) that can be managed using IP-plan definitions.

Site
The Site is a location of a Client. Each Site has a Site_type associated that defines what Service_classes can be configured.

Service
Any Site can have many Services within the restrictions imposed by the Service_class. A NetYCE Service is a container that groups objects like Nodes and IP Subnets (v4/v6) that deliver (part of) a service.

Node
All Nodes (network devices) reside in a Service container. It can only be created using a Service-type which in turn relies on a Node_type definition. Node_types are grouped in the modelling using Node_classes. The node-type defines many (initial) attributes like the Template and the Hostname. The Template also determines the vendor, its hardware and the port-blueprint. A Node has many attributes but also links into its Domain, the Ports and their Topology. Optionally the VRF's and eVPN's it uses.

Port
Ports are assigned to Nodes and are assigned a Port-template that defines their function. Ports can be linked using topology. The Port object manipulates just one Port, the Ports object a series of ports.

Ports
Ports are assigned to Nodes and are assigned a Port-template that defines their function. Ports can be linked using topology. The Port object manipulates just one Port, the Ports object a series of ports.

Link
A Link is technically a set of two Ports sharing Topology. A link can be used to assign shared attributes or to locate the uplink or downlink node.

Slot
Again, a Slot is a series of ports that can be manipulated as a group, but in this case they share a common slot-id. Slots are mostly used when moving ports between nodes or dealing with stack configurations.

Supernet
Each Client can be allocated one or more IP Supernets (IPv4 and IPv6) that it can use exclusively to extract new IP Subnets from. A Supernet must have an IP-plan assigned that defines how the Supernet may be used to extract the various Subnet-types from. In the NetYCE IP modelling, all subnets are referenced using their name as defined in the IP-plan. The IP-subnet-plan then defines which and how the subnet variables (like Vlan-id) are determined. If no IP-plans can be used (due to non-systematical issuing of ranges), Supernets serve no purpose and custom Subnets must be used.

Subnet
Two types of IPv4 subnets exist in NetYCE, those that conform to an IP-plan and IP-subnet-plan, and those that do not. The latter type use the 'Custom' format for which the various attributes are free to

alter. IP-plan based subnets are fully defined in the subnet-plan and cannot be modified. Subnets, like Nodes can only reside in a Service container (thereby determining the Service-type). In the NetYCE IP modelling, all subnets are referenced using their name as defined in the IP-plan. The IP-subnet-plan then defines which and how the subnet variables (like Vlan-id) are determined.

Address
Subnets are basically a group of IPv4 addresses. Specific addresses can be located and allocated within a subnet or within ranges of a subnet.

Ipv6_net
Two types of IPv6 subnets exist in NetYCE, those that conform to an IP-plan and IP-subnet-plan, and those that do not. The latter type use the 'Custom' format for which the various attributes are free to alter. IP-plan based subnets are fully defined in the subnet-plan and cannot be modified. IPv6 subnets, like Nodes can only reside in a Service container (thereby determining the Service-type). In the NetYCE IP modelling, all subnets are referenced using their name as defined in the IP-plan. The IP-subnet-plan then defines which and how the subnet variables (like Vlan-id) are determined.

Ipv6_addr
IPv6 Subnets are basically a group of IPv6 addresses. Specific addresses can be located and allocated within a subnet or within ranges of a subnet.

Domain
Every Node has a reference to a Domain. The Domain maintains a large number of attributes that combined define the Network Management environment for that node. Most importantly, it has the user-ids and passwords of the functional users that NetYCE will use to login into the device. Many different Domains can exist within the various networks, but a Node can only be a member of one Domain.

Server
Although not strictly a Networking component, Servers and their IP-addresses can have an important role in the various configurations. Often a Client or Site requires a server-related entry that may use addresses or hostname. Servers can also be used to represent access-list entries. NetYCE supports IPv4 and IPV6 or dual-stack based Servers.

Mpls_vrf
For VRF definitions and assignments, NetYCE uses two object types, Mpls_vrf and Vrf. The Mpls_vrf object defines which VPN's exist on 'the' MPLS cloud and how the Route-distinguisher and Route-targets are formed for each of the Client-types that is allowed to use the VPN. The Vrf object forms the assignment of a Node to a Mpls_vrf and uses calculated values for the RD and RT.

Vrf
For VRF definitions and assignments, NetYCE uses two object types, Mpls_vrf and Vrf. The Mpls_vrf object defines which VPN's exist on 'the' MPLS cloud and how the Route-distinguisher and Route-targets are formed for each of the Client-types that is allowed to use the VPN. The Vrf object forms the assignment of a Node to a Mpls_vrf and uses calculated values for the RD and RT.

Region
Every Client is a member of a Region. The Region defines common variables for a series of Clients, regardless of Client_type. Regions use only Custom Attributes.

Alias names

Alias names in Service-types have only local significance. They do not exists outside the running Service-type. So they can have any name you like and nothing will prevent you.

However it is recommended to adopt a naming convention that will be used by all designers creating Service-types. Especially with longer Service-types, many aliases will likely be created, which for the occasional editor may not be easily recognizable as such. If alias names start looking like hostnames or subnet names, how to tell them apart?

For this reason, we recommend to enclose alias names between carets (‘<‘ and ‘>’). This strongly associates with the variable usage in Templates and Scenarios.

Another recommendation is the inclusion of the ’Type’ of the alias. It is not needed since NetYCE knows and checks the type of each alias during the syntactical check before the Service-type starts, but it can greatly improve the readability of your Service-type.

Aliases like <service> and <node> is only sufficient for the very simplest of Service-types. As soon as more than one is involved, more descriptive names are helpful: <core.srv> and <core.node>.

Especially when linking up nodes with multiple ports and portchannels is a naming convention essential. For example linking up two core nodes with dual links and a portchannel could require these aliases:

```
<core.A.srv> <core.A.node> <core.A1.port> <core.A2.port> <core.Apo.port>
<core.B.srv> <core.B.node> <core.B1.port> <core.B2.port> <core.Bpo.port>
<core.AB1.link> <core.AB2.link> <core.ABpo.link>
```

Or an access switch linking up to a redundant set of core switches:

```
<core.srv> <coreA.node> <coreA.downlink.port>
<core.srv>  <coreB.node>  <coreB.downlink.port>
<access.uplinkA.port> <access.uplinkB.port>
<coreA.access.link> <coreB.access.link>
<access.mgmt.net> <access.mgmt.addr> <access.mgmt.port>
<access.user.net> <access.ipt.net>
```

The following list of alias types could be useful:

```
client - client (customer)
sit    - site (location)
srv    - service (container)
node   - node (device)
dom    - domain (management)
port   - port (interface)
ports  - ports (set of interfaces)
link   - topology (2 ports)
slot   - module (interfaces using slotid)
```

```
srv    - server
super  - supernet
net    - ipv4 subnet
addr   - ipv4 address
v6net  - ipv6 subnet
v6addr - ipv6 address
vpn    - mpls-vrf
vrf    - vrf
```