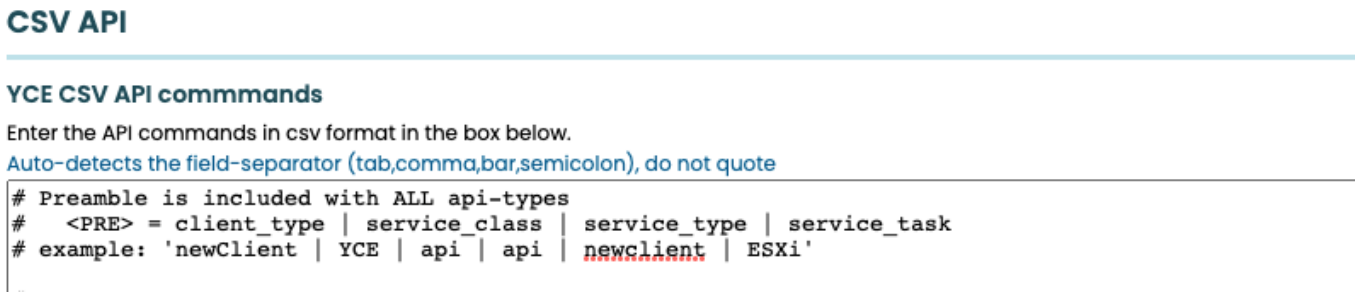
Table of Contents

CSV API

The NetYCE CSV API is a tool to simplify the use of the API (*application programming interface*) for Service-types. It acts as a “wrapper” to create and execute NetYCE Service-type API xml-calls calls using simple CSV formatted requests.

The CSV API can only be used within the NetYCE front-end tool. It is therefore not a true API that can be accessed externally, the NetYCE XML based API is intended for those purposes.



It is feasible to extend the xml API with a function that will accept these CSV formatted lines in the XML request and process them as this tool does. If requested, NetYCE will build this function for our customers.

Setting up

Service-tasks

The CSV API requires some setup. To start with, it requires the service-types/tasks to exist in order to execute them. And, these service-tasks need to be designed to accept the variable names that the API call includes. It is a good idea to create service-tasks that are specifically created to be used using the API and are not accessible for generic use in the NetYCE GUI.

Also, the service-types and tasks that are created for generic (i.e. GUI) usage are normally designed to execute a complete change or service, a single service-task that will do-it-all in one go. Consequently these tasks can become quite lengthy. A Service-task consisting of over 40 service calls is not uncommon.

That approach is not desirable for API usage. It is better to create a couple of shorter, simpler service-tasks, each performing a specific step in creating a service or executing a change. Combining these shorter tasks in a sequence of API calls is quite the thing to do when using an API.

The principal reason for creating simple and short service-task is to simplify variable handling. Executing an extensive task can require a lot of variables that need to be included in the one API call. By splitting these up, each API service-task may need only a few, simplifying selecting the variables to include in the call and making it more obvious where these variables are used in the service-task.

The CSV API is intended to work in such a set-up: A series of simple CSV lines, each calling on a rather basic service-task that requires only a few variables.

CSV definitions

The CSV lines entered in the CSV API tool must directly map onto the service-task(s) it will execute. Each CSV line will therefore consist of three parts:

1. The name of the CSV line-type.
2. The service-task identifier.
3. The list of variables this CSV line-type supports.

The CSV line type name tells the CSV API which set of variables it must expect to find in the CSV line, and to what variable names these values must be assigned to. Some of these variables are mandatory, others are optional and may be blank.

The service-task identifier consists of four parts: The *Client_type*, the *Service_class*, the *Service_type* and *Service_task*. All four fields must be present and the corresponding service-task should exist in the NetYCE modelling. These are referred to by <PRE> in the CSV API commands.

Following these identifier fields, the list of variable values are appended. The number of values and what variable names they will be assigned depends on the CSV line-type and its definition. Although fully customizable and extensible, a default set of line types is preconfigured.

When accessing the CSV API tool, the area where the CSV lines will be typed or pasted into, will list the configured CSV line types along with a brief explanation.

Field Separators

The CSV API tool can autodetect the separator used (within that line). the supported field separators are *comma*, *tab*, *semicolon* and the *vertical bar* symbols.

Values should NOT be quoted and white space around values will be automatically stripped. Values are NOT case sensitive.

For maximum readability it is recommended to use the | as field separator insert a space on either side.

Default CSV line types

The default preconfigured CSV line types include `newClient`, `newSite`, `newSrv`, `addNode` and some others. These should give the user a basic idea of the kind of service-tasks suitable for the CSV API.

```
# Preamble is included with ALL api-types
#   <PRE> = client_type | service_class | service_type | service_task
# example: 'newClient | YCE | api | api | newclient | ESXi'

# newClient
# Add a (new) client to the network
# newClient | <PRE> | client_code | [client_name]
```

```
# newSite
# Add a (new) site to a client
# newSite | <PRE> | client_code | site_type | site_code

# newSrv
# Create new service on location. Use specified service_name or default
# newSrv | <PRE> | client_code | site_code | [service_name]

# addNode
# Add node to service. Use service_name to locate or create the service for
the new node
# addNode | <PRE> | client_code | site_code | node_type | node_name |
[service_name]

# addCnet
# Add custom subnet. Use node_name to locate the service for this subnet
# addCnet | <PRE> | node_name | net_name | net_address | net_prefix |
[vlan_id] | [vlan_tpl] | [vrf_name]

# addSupernet
# Add a supernet of a given ip-plan to the client
# addSupernet | <PRE> | client_code | ip_supernet | ip_plan | [dns_domain]

# addNet
# Add ip-plan-based subnet. use node_name to locate the service for this
subnet
# addNet | <PRE> | node_name | net_name | [vlan_tpl] | [vrf_name]

# addLink
# Create topology link between two ports and reconfigure those
# addLink | <PRE> | node_nameA | port_nameA | node_nameB | port_nameB |
[port_shut] | [port_speed] | [port_mode] | [port_chanA] | [port_chanB]

# serviceTask
# Execute a service task with between 2 and 6 nodes as input
# serviceTask | <PRE> | node_nameA | node_nameB | [node_nameC] |
[node_nameD] | [node_nameE] | [node_nameF]
```

Defining CSV line-types

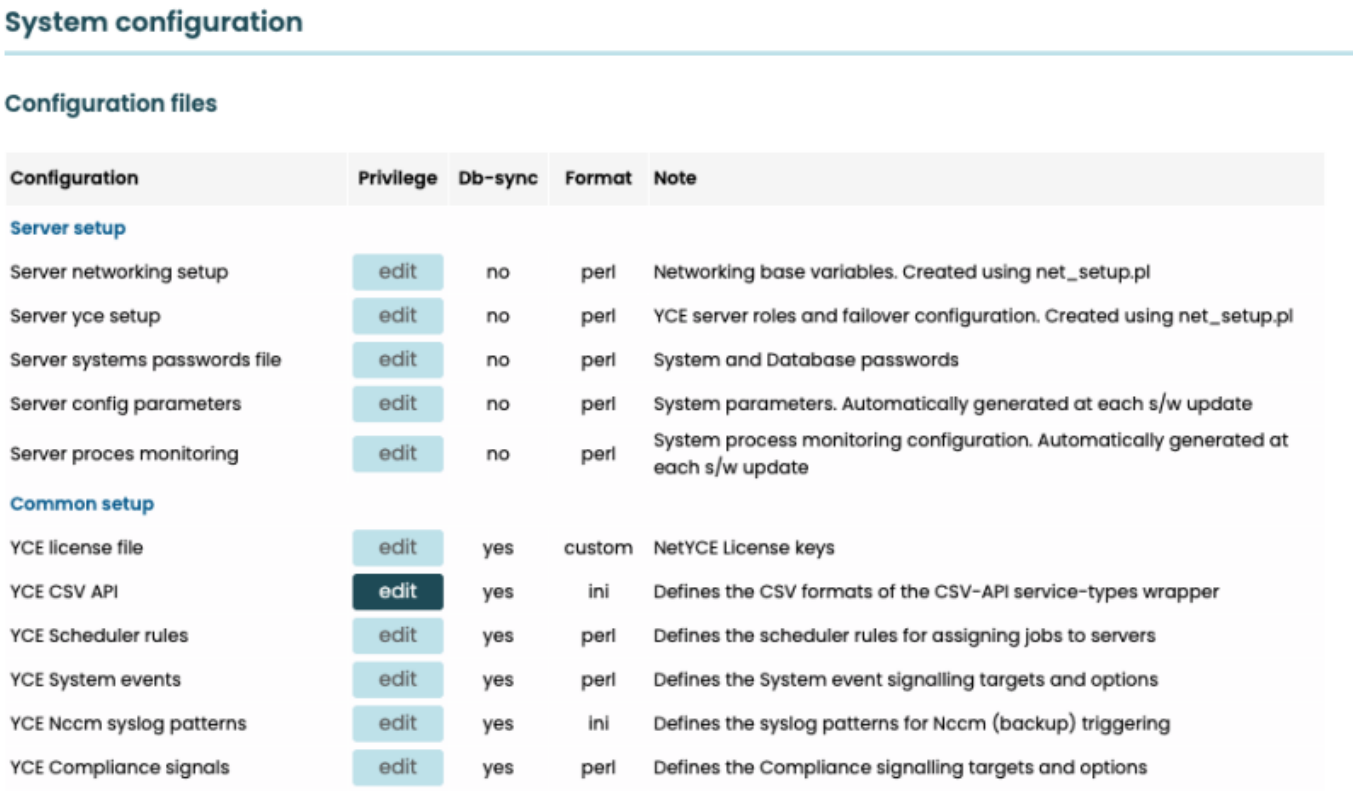
The default CSV line types can easily be expanded to create CSV line types more specific to a network design. The samples below may illustrate this:

```
# addCPE
# Create new IPVPN customer CPE including many custom vars
# addCPE | <PRE> | Customer_name | Customer_site | [Srv_type] |
[PTSID_service] | [RD] | [VRF_name] | CPE_hostname | CPE_type | CPE_template
| [CPE_port] | [DB_id] | [Mpls_neighbor] | [Ring_hostname] | [Ring_port] |
[Ring_vlan] | [VPN_id] | var
```

```
# newRing
# Start new "ring" between the PE's on the site, name it 'service_name',
create link as start
# newRing | <PRE> | client_code | site_code | service_name | A_ring_port |
B_ring_port

# ringInsert
# ringInsert | <PRE> | client_code | site_code | Ring_name | Ring_node |
A_node | B_node | RtoA_port | RtoB_port
```

To add these line types, access the “Edit Configs” tool from the “Admin - System” menu and edit “YCE CSV API” configuration file.



System configuration

Configuration files

Configuration	Privilege	Db-sync	Format	Note
Server setup				
Server networking setup	edit	no	perl	Networking base variables. Created using net_setup.pl
Server yce setup	edit	no	perl	YCE server roles and failover configuration. Created using net_setup.pl
Server systems passwords file	edit	no	perl	System and Database passwords
Server config parameters	edit	no	perl	System parameters. Automatically generated at each s/w update
Server proces monitoring	edit	no	perl	System process monitoring configuration. Automatically generated at each s/w update
Common setup				
YCE license file	edit	yes	custom	NetYCE License keys
YCE CSV API	edit	yes	ini	Defines the CSV formats of the CSV-API service-types wrapper
YCE Scheduler rules	edit	yes	perl	Defines the scheduler rules for assigning jobs to servers
YCE System events	edit	yes	perl	Defines the System event signalling targets and options
YCE Nccm syslog patterns	edit	yes	ini	Defines the syslog patterns for Nccm (backup) triggering
YCE Compliance signals	edit	yes	perl	Defines the Compliance signalling targets and options

Scroll to the end of the file and add the following lines:

```
[newRing]
brief = Start new "ring" between the PE's on the site, name it
'service_name', create link as start
client_code
site_code
service_name
A_ring_port
B_ring_port

[ringInsert]
client_code
site_code
Ring_name
Ring_node
A_node
```

```
B_node
RtoA_port
RtoB_port

[addCPE]
brief = Create new IPVPN customer CPE including many custom vars
Customer_name
Customer_site
Srv_type = opt
PTSID_service = opt
RD = opt
VRF_name = opt
CPE_hostname
CPE_type
CPE_template
CPE_port = opt
DB_id = opt
Mpls_neighbor = opt
Ring_hostname = opt
Ring_port = opt
Ring_vlan = opt
VPN_id = opt
```

Each line type definition starts with a section header where the line type name is set between square brackets: `[csv_name]`

The next line includes a brief description of the line type. All subsequent lines define the variables that the line-type requires. By default the variable is **mandatory**, optional variable require the `= opt` to be appended.

Mandatory variables demand a value to be present at run-time. A blank value will cause the CSV API to issue an error message when parsing the CSV command lines.

Please see the article on Edit Configs how to use this tool.

Defining API Service-tasks

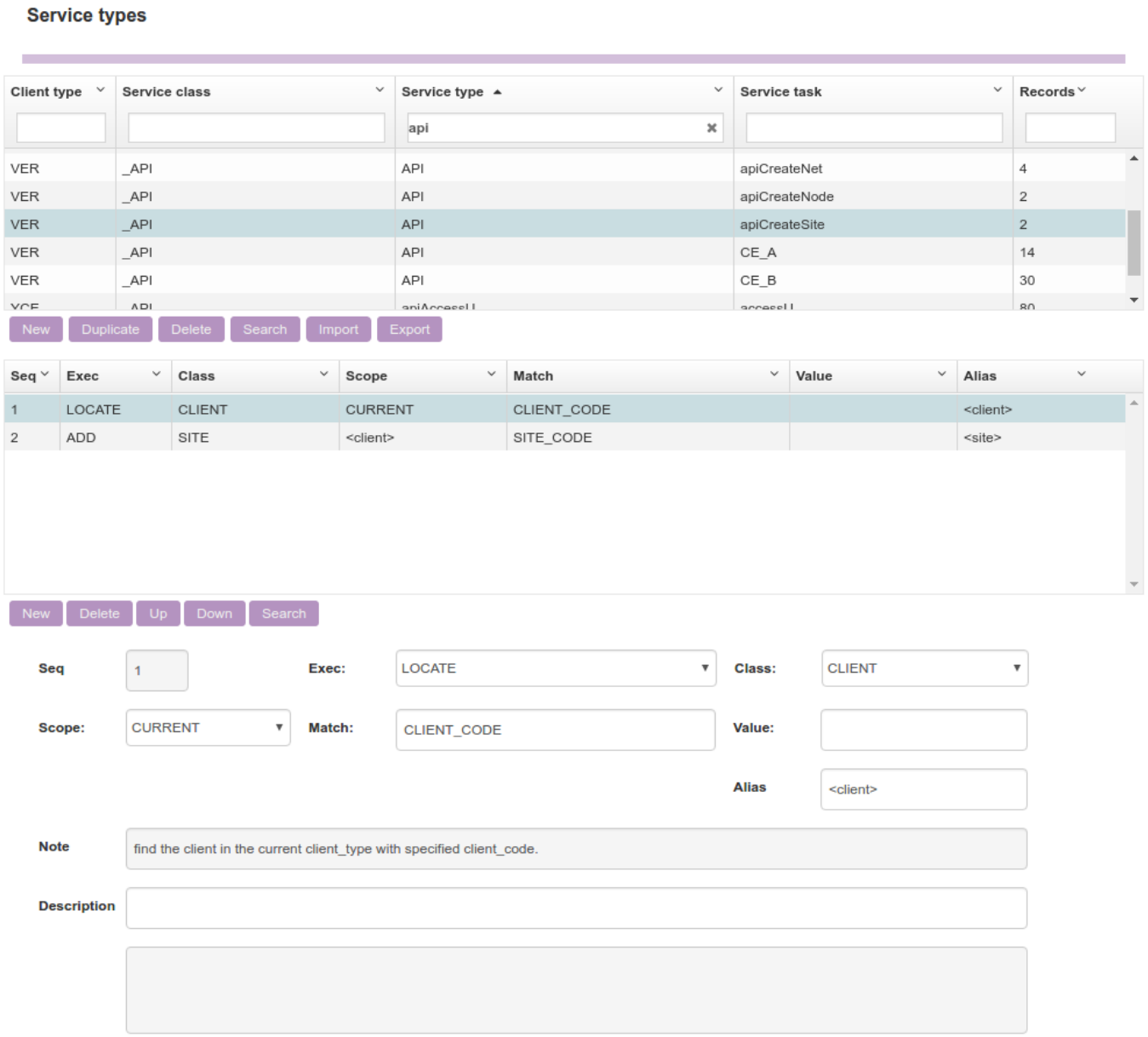
Although creating the API service tasks are straightforward, a few practical pointers and examples will be very helpful for the frist-time user.

Since we aim to keep the API service tasks simple and short, they are likely to be reusable in a number of situations. Therefore, it is recommended to group all API service tasks together in the same service_class. We often use `api` for this purpose. This service_class can be created as part of any of the Site_types or in a Site_type by itself, it makes no difference.

When API service-tasks are more specific to a Service_class, then it is wise NOT to use the generic class 'api' but create it using the Service_class it belongs to. Nevertheless it is never a good idea to use an existing Service_type name to store the API call: It will then become available in the Service-details' 'add' form.

Be aware that the API service-tasks are specific to the Client_type. API service-tasks cannot be shared

across Client_types. The operators permissions for the Client_type also apply when executing the CSV API.



In the example above, the Service_class `api` was created and used for two tasks. Both are using the generic `api` Service_type and use the Service_task name matching the CSV line-type. Using the CSV line type can be useful, but is not a dependency. These API tasks will be executed by the API, the CSV API is just a wrapper assisting to create the actual API calls. The line type name is irrelevant to the API.

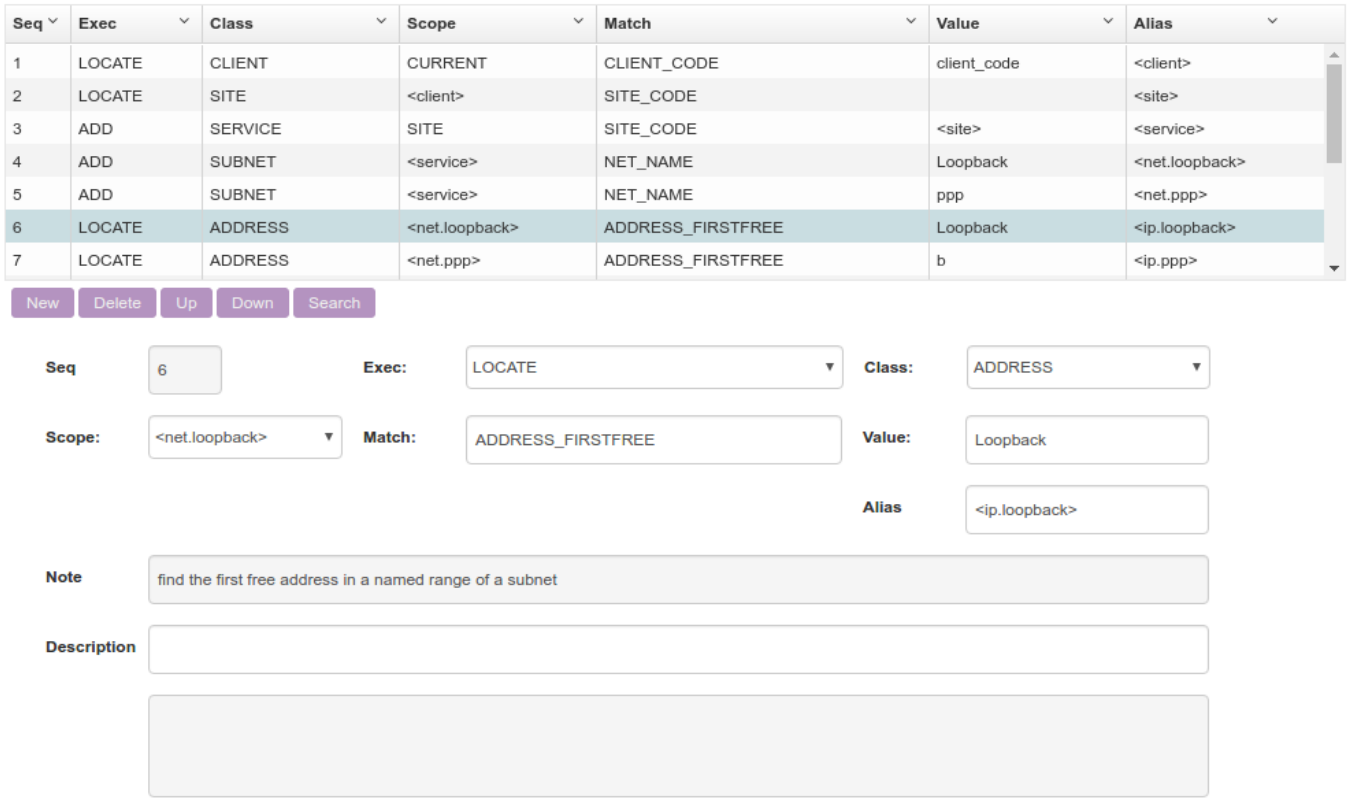
The examples show two very, very basic tasks. The `newClient` executes just one service type call: “ADD - CLIENT - CLIENT_CODE”. And the `newSite` just two: “LOCATE - CLIENT - CLIENT_CODE” and “ADD - SITE - SITE_CODE”

The really only notable thing here is how the variable names from the API call are being used in these service type calls. The API variable name is entered in the Value field using round brackets or parentheses, `(` and `)`.

If this variable name is present and the API call its value will be used, otherwise the default value will be assumed. It is mostly this use of variables from the API that sets these service-tasks apart from the

ordinary GUI versions.

The following example shows the service task `addAccessNode`. It shows one of many possible implementations, all dependent on the architecture and modelling involved. In this case the Node_type and Node_name are variables but the loopback address is lifted from an Ip-plan offering the 'Loopback' subnet type.



Seq	Exec	Class	Scope	Match	Value	Alias
1	LOCATE	CLIENT	CURRENT	CLIENT_CODE	client_code	<client>
2	LOCATE	SITE	<client>	SITE_CODE		<site>
3	ADD	SERVICE	SITE	SITE_CODE	<site>	<service>
4	ADD	SUBNET	<service>	NET_NAME	Loopback	<net.loopback>
5	ADD	SUBNET	<service>	NET_NAME	ppp	<net.ppp>
6	LOCATE	ADDRESS	<net.loopback>	ADDRESS_FIRSTFREE	Loopback	<ip.loopback>
7	LOCATE	ADDRESS	<net.ppp>	ADDRESS_FIRSTFREE	b	<ip.ppp>