NSO Cisco PnP Core Function Pack 3.0.0

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1

NSO Cisco PnP Guide

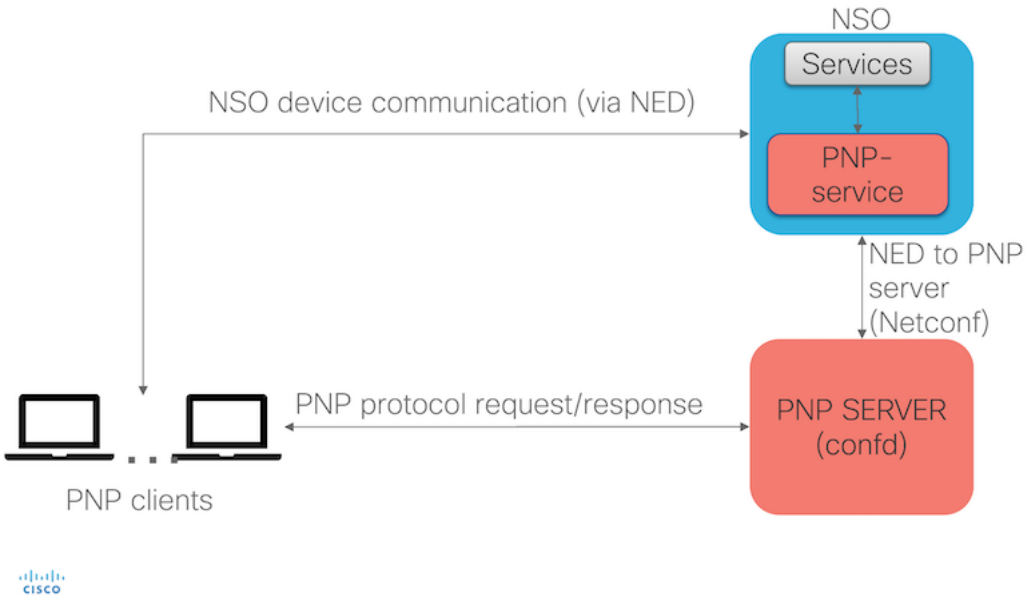
Introduction

The NSO Plug and Play (PnP) Core Function Pack (CFP) enables provisioning and managment of Cisco PnP clients according to the Cisco PnP Protocol. This allows a secure and scalable solution for applying Day-0 configurations to compliant Cisco devices and the option for managing those devices from NSO.

Overview

The PnP CFP consists of a PnP Service package on NSO along with one or more PnP Servers running on instances of ConfD. PnP Client Devices will contact a PnP Server on ConfD, which will then update the NSO PnP Service. The PnP Service definition for the device determines the Day-0 configuration that will be pushed to the device.

Figure 1. NSO PnP CFP Architecture



Installation

Installation is a two-step process, first ConfD must be installed along with the PnP Server and then the NSO PnP Service must be set up. The PnP Service package can run on a new NSO installation or as part of an existing runtime environment.

Procedure 1.1. Instructions on how to install

Step 1 **Download and Run the PnP Installer**

The PnP CFP can be downloaded from Cisco CCO as a signed binary. The binary contains the installer for the PnP Server, which includes ConfD, as well as a `core-fp-packages` folder containing the NSO PnP Service packages. These two packages are the PnP Service packages itself and the NED used to communicate with the ConfD server.

```
$ sh nso-5.3.1.1-nso-pnp-3.0.0.signed.bin
$ tar –xvf nso-5.3.1.1-nso-pnp-3.0.0.tar.gz
$ cd ~/nso-5.3.1.1-nso-pnp-3.0.0/
$ ls
core-fp-packages  pnp-3.0-installer.bin
$ cd core-fp-packages/
$ ls
ncs-5.3.1.1-cisco-pnp-3.0.0.tar.gz  ncs-5.3.1.1-confd-pnp-ned-nc-1.0.tar.gz
```

Step 2 **Install the PnP Server**

Navigate to the directory containing the `pnp-3.0-installer.bin` file and perform a system installation.

Note This will also install ConfD

```
$ cd ~/nso-5.3.1.1-nso-pnp-3.0.0/
$ sudo sh pnp-3.0-installer.bin –-system-install
INFO  Using temporary directory /tmp/pnp_installer.27501 to stage PNP installation bundle
INFO  Using /opt/pnp-server/pnp-3.0.0 for static files
```

```
INFO  Using /etc/pnp-server for configuration files
INFO  Using /var/opt/pnp-server for run-time state files
INFO  Using /var/log/pnp-server for log files
INFO  Unpacked confd-7.3 in /opt/pnp-server/pnp-3.0.0
INFO  Generating default SSH hostkey (this may take some time)
INFO  SSH hostkey generated
INFO  Environment set-up generated in /opt/pnp-server/pnp-3.0.0/confdrc
INFO  ConfD installation script finished
INFO  Generating self-signed certificates for HTTPS
INFO  Installed init script /etc/init.d/confd
INFO  Installed user profile script confd.sh in /etc/profile.d
INFO  Installed user profile script confd.csh in /etc/profile.d
INFO  PNP server installation complete
```

The ConfD server can now be started with `sudo /etc/init.d/confd start`. Be sure to verify the `confd.xml` settings and to source either `/etc/profile.d/confd.sh` or `/etc/profile.d/confd.csh`

Step 3 **Install the PnP Service and ConfD NED Packages**

Next, install the PnP Service package `ncs-5.3.1.1-cisco-pnp-3.0.0.tar.gz` and the NED for communicating with the PnP Server `ncs-5.3.1.1-confd-pnp-ned-nc-1.0.tar.gz` into either an existing NSO server or a new NSO installation.

Note The NSO must be or have been installed using system install

The following commands show an example of loading the packages in NSO, first they are copied to `/opt/ncs/packages` then soft linked to the NSO packages directory:

```
$ sudo cp /home/admin/nso-5.3.1.1-nso-pnp-3.0.0/core-fp-packages/
  ncs-5.3.1.1-confd-pnp-ned-nc-3.0.tar.gz /opt/ncs/packages/.
$ sudo cp /home/admin/nso-5.3.1.1-nso-pnp-3.0.0/core-fp-packages/
  ncs-5.3.1.1-cisco-pnp-3.0.0.tar.gz /opt/ncs/packages/.
$ cd /var/opt/ncs/packages/
$ sudo ln -s /opt/ncs/packages/ncs-5.3.1.1-confd-pnp-ned-nc-1.0.tar.gz \
  ncs-5.3.1.1-confd-pnp-ned-nc-1.0.tar.gz
$ sudo ln -s /opt/ncs/packages/ncs-5.3.1.1-confd-pnp-ned-nc-1.0.tar.gz \
  ncs-5.3.1.1-cisco-pnp-3.0.0.tar.gz
$ sudo /etc/init.d/ncs restart-with-package-reload
```

After the two packages have been installed verify they are operational:

```
admin@ncs> show packages package package-version
                     PACKAGE
NAME                 VERSION
------------------------------
confd-pnp-ned-nc-1.0  1.0
pnp-service          3.0.0

admin@ncs> show packages package oper-status up
NAME                 UP
-------------------------
confd-pnp-ned-nc-1.0  X
pnp-service          X
```

Step 4 **Setup Access Control Rule for NSO**

Before a PnP Server can be onboarded into NSO the proper `nacm` rule must be set for NSO. The NSO username must be added to the `ncsadmin` group in the `nacm` rule in order to trigger re-deploys automatically. See below for example commands:

```
admin@ncs> configure
```

```
admin@ncs% show nacm groups group ncsadmin
user-name [ private ];
admin@ncs% set nacm groups group ncsadmin user-name [ admin private ]
admin@ncs% commit
Commit complete.
admin@ncs% show nacm groups group ncsadmin
user-name [ admin private ];
```

Note See the NSO documentation for further information on how to add and configure users and permissions.

Step 5 **Onboard PnP Server Device to NSO**

Follow the standard NSO procedure to onboard and sync the PnP Server ConfD device.

```
$ ncs_cli –u admin
admin@ncs> configure
admin@ncs% set devices authgroups group pnp default-map remote-name admin remote-password admin remote-se
admin@ncs% set devices device confdpnp address 192.168.66.53 port 2022
admin@ncs% set devices device confdpnp authgroup pnp
admin@ncs% set devices device confdpnp device-type netconf ned-id confd-pnp-ned-nc-1.0
admin@ncs% set devices device confdpnp trace pretty
admin@ncs% set devices device confdpnp state admin-state unlocked
admin@ncs% commit

admin@ncs% request devices device confdpnp ssh fetch-host-keys
result updated
fingerprint {
    algorithm ssh-rsa
    value 8c:c5:2d:6c:40:52:6b:51:46:9e:a4:f4:66:5e:74:ac
}
admin@ncs% request devices device confdpnp connect
result true
info (admin) Connected to confdpnp - 192.168.66.53:2022
admin@ncs% request devices device confdpnp sync-from
result true
admin@ncs% request devices device confdpnp check-sync
result in-sync
admin@ncs%
```

Note The IP address and port to use during onboarding can be found in the `/etc/pnp-server/confd.conf` file on ConfD. Check the `netconf-north-bound` section for the `ssh` port.

Uninstall

To perform a clean uninstall of the PnP CFP use the following procedure.

Step 1 **Delete PnP Map**

The following command shows deleting the PnP map with the NSO cli:

```
$ ncs_cli –u admin
admin@ncs> configure
admin@ncs% delete pnp map
admin@ncs% commit
```

Step 2 **Delete PnP Devices**

The following commands show deleting the PnP devices with the NSO cli:

```
admin@ncs% delete devices device XXXX
admin@ncs% commit
```

Step 3 **Remove PnP NSO Packages**

The following commands show removing the PnP packages with the NSO cli:

```
admin@ncs> request software packages deinstall package pnp-service
admin@ncs> request software packages deinstall package confd-pnp-ned-nc-1.0
admin@ncs> request package reload
```

Step 4 **Uninstall the PnP ConfD Server**

```
$ source /etc/profile.d/confd.sh or source /etc/profile.d/confd.csh
sudo –s
# pnp-uninstall ––all –non-interactive
# exit
$ cd ~
```

Note The `confd.sh` or `confd.csh` files must be sourced.

CHAPTER 2

NSO Cisco PnP Server

Overview

The PnP Server runs on a ConfD instance and handles all PnP traffic to and from the device, described below. The server is configured from the NSO PnP Service covered in the next chapter. There can be one or more instances of the PnP Server managed by NSO as scaling demand requires.

PnP Server Discovery

The typical PnP Client has a day--1 (day minus one) configuration with not much more than instructions to contact a PnP Server for the day-0 configuration. To be able to find the PnP Server the client uses *PnP Discovery*.

Typically DNS is used for PnP Discovery, therefore the DNS server needs to be configured with an entry for the NSO PnP Server. The exact configuration of the DNS server depends on the type of the DNS server.

For example: **ip host pnpserver.xyz.com 10.30.30.10**

The PnP Client will then initiate the **PnP HELLO** sequence where it starts to communicate with the PnP Server.

PnP Server HTTP(s) interface endpoints

The PnP Server exposes a number of HTTP URL:s used in the PnP protocol. These are not configurable and are available either on the primary, using either HTTP or HTTPS, or the secondary, only HTTP, server port.

Table 2. Resources

Resource	HTTP Method	Description
/pnp	POST	The root resource and PnP protocol endpoint

Resource	HTTP Method	Description
/pnp/HELLO	GET	Used by the PnP Agent in Server discovery
/pnp/day0/<device serial here>	GET	Every device with a configured serial, can GET the day0 configuration from here. If config-apply is enabled.
/pnp/ca/ca.cert	GET	When the PnP Server uses a self-signed certificate, the device can GET the ca.cert from this endpoint in order to upgrade the session to HTTPS.

PnP Server Logging

There are several log files that may be of interest. All logs exist under the ConfD `var/log/pnp-server` directory.

`pnpserver:<port>.access` A work request access log. When a PnP Client device contacts the PnP Server there will be entries such as:

```
127.0.0.1 - - [date] "POST /pnp/HELLO HTTP/1.0" 200 0 "-" "-"
127.0.0.1 - - [date] "POST /pnp/WORK-REQUEST HTTP/1.0" 200 234
127.0.0.1 - - [date] "POST /pnp/WORK-RESPONSE HTTP/1.0" 200 209
```

`pnpserver:<port>.auth` A web log containing all authentication related messages

`PID%%%%.trace` A trace file for a specific PnP Client. The file name will depend on the serial number of the device. These log files depend upon the logging configuration received from the PnP Service. See the PnP Service chapter for further details.

`devel.log` The Erlang application log

`confd.log` The ConfD log file

`confderr.log` The ConfD error log file

CHAPTER 3

NSO Cisco PnP Service Usage

Overview

The NSO PnP Service is the main interface for users of the CFP. After the PnP Servers have been installed and setup all user interaction will be done from this service. This service is responsible for configuring each PnP Server as well defining a map to each client device to configure such things as the authentication strategy and the Day-O configuration sent to the device.

PnP Server Configuration

The PnP Service sets the configuration for each PnP Server, defining how the server will listen to and handle client communication, security, if certificates will be enabled, and logging. This data is set in the `pnp/servers` list for a specific onboarded PnP Server.

server

The `server` container defines the IP address or addresses and port on which to listen for incoming requests, as well as whether to enable HTTPS, and if certificate authentication will be enabled. Some important settings that can be configured here are:

ip-address	The address on which the server will listen for incoming requests.
additional-ip-addresses	Additional addresses on which the PnP server will listen for incoming requests.
port	The port on which the server will listen for incoming requests.
use-ssl	If set to **`true`** will enable HTTPS.
verify-cpe-cert	If set to **`true`** clients will require authentication.
ca-cert-file	The certificate file to use when authenticating a client device.

Tip Consult the `tailf-ned-cisco-pnp.yang` model or use the NSO cli for the full descriptions and default values.

Below is a sample that configures the server to listen on all addresses, changes the default port from 9191 to 9192, and tells the server to use HTTPS instead of HTTP:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# pnp server ip-address 0.0.0.0
admin@ncs(config)# pnp server port 9192
admin@ncs(config)# pnp use-ssl true
admin@ncs(config)# pnp verify-cpe-cert false
admin@ncs(config)# commit
```

Note When using HTTPS the same certificate as used for NSO will be used in the PnP HTTPS server.

This section is important for SUDI.

proxy-servers

The `proxy-servers` container is used when a proxy server is used in front of the PnP Server. The user can set whether `x-forwarded-for` is set to always take the client IP from the header or to only honor from specific IP addresses.

secondary-server

The `secondary-server` container is used for certificate installation. It allows for configuration of the `ip-address`, `additional-ip-addresses`, and `port` to listen for incoming requests.

state-value

The `state-value` container is where aspects of how the server communicates with the PnP clients is configured. The following settings are available:

ca-cert-file	The certificate file to use when authenticating a client device.
backoff-timeout	Client devices will repeatedly ask the PnP Server for any configuration updates. If there are no updates to send the server will tell the device for the given amount of seconds defined here.
terminate-when-done	If present this indicates to the server that the client will no longer re-sync after onboarding.

device-map

The `device-map` is a convenient way to apply a group of settings across many client devices. It uses a regular expression to match against the `device-info` received from the PnP client. Some of the values that can be defined are: the WAN and LAN interfaces of the device, the `trustpoint-label`, and whether `config-restore` is done from a file, URI, or disabled.

Tip The list above is not comprehensive, consult the `tailf-ned-cisco-pnp.yang` model or use the NSO cli for the full list and descriptions.

logging

The `logging` container defines a `serial` list where users can specify specific PnP client serial numbers for which the PnP Server will trace the communication with the device. If **`all`** is entered instead of a serial number then the server will create a trace log for all PnP clients.

```
admin@ncs(config)# pnp logging serial all
admin@ncs(config)# commit
```

wait-after-reload-time

This leaf controls the time to wait, in seconds, between issuing a reload and applying the Day-0 configuration for a client device.

PnP Map

The `pnp/map` is an NSO service used to define how to configure and interact with the PnP client. It determines the authentication method, the Day-0 that will be pushed to the device, and also has elements to exercise features of the PnP protocol.

Plan

Each entry in the `pnp/map` will have an associated NSO plan showing the current state of the service. There are two versions of the plan depending upon whether the device is managed or unmanaged. The plan can be accessed at `pnp/map-plan` from NSO in operational mode.

Note See the NSO Developer's Guide for more information about nano services and plans.

Unmanaged Devices

An unmanaged device is one that will not be added to the NSO device tree. These devices are indicated by having the `managed` leaf in the `pnp/map` set to false.

The example below shows the plan for an unmanaged device where the device has yet to contact the PnP Server. Notice that the `device-connected` state and the `unmanaged-device ready` state are both `not-reached`.

```
admin@ncs> show pnp map-plan pios-2
                        BACK
TYPE              NAME    TRACK  GOAL  STATE             STATUS       WHEN
-------------------------------------------------------------------------------------
self              self    false  -     init              reached      2020-03-18T18:57:20
                                       device-connected  not-reached  -
                                       ready             not-reached  -
unmanaged-device  pios-2  false  -     init              reached      2020-03-18T18:57:20
                                       ready             not-reached  -
```

Managed Devices

A managed device is one that will be added to the NSO device tree and further managed by NSO. These devices are indicated by having the `managed` leaf in the `pnp/map` set to true.

A managed device plan has a TYPE of `managed-device` and some additional plan states reflecting the device onboarding into NSO. The following example shows the plan for a managed device that has contacted the PnP Server. Note that both the `device-connected` and `managed-device ready` states are `reached`.

```
admin@ncs> show pnp map-plan pios1
                     BACK
TYPE            NAME   TRACK  GOAL  STATE             STATUS   WHEN
-------------------------------------------------------------------------------
self            self   false  -     init              reached  2020-03-18T19:08:12
                                     device-connected  reached  2020-03-18T19:08:12
                                     ready             reached  2020-03-18T19:08:12
managed-device  pios1  false  -     init              reached  2020-03-18T19:08:12
                                     device-created    reached  2020-03-18T19:08:15
                                     fetch-host-keys   reached  2020-03-18T19:08:15
                                     sync              reached  2020-03-18T19:08:16
                                     ready             reached  2020-03-18T19:08:16
```

Tip For services utilizing the PnP CFP that need to react to events about the lifecycle or status of a PnP client device they can set up a `kicker` on any of the `plan` component states. See the NSO documentation for further iformation on using kickers.

Operational Data

The PnP Service maintains operational data about the PnP client devices that have contacted the PnP Server. This includes data about the state of the device as well as a serial history. The data is updated via notifications to NSO from the PnP Server.

PnP State

The operational `state` list is saved based on the serial number for every mapped device. It keeps information received from the device such as the UDI and device-info, as well as internal state data such as the claimed status and last-event-type received.

The following is an example of the state for a device with serial number PIOS-1. Note that the `claimed` leaf is currently `true` indicating the device has been claimed by a PnP service. If this value were `false` it would mean that the device had contacted the PnP Server, but there was not yet an entry for it in the `pnp/map`.

```
admin@ncs> show pnp state PIOS-1
pnp state PIOS-1
 udi               PID:CISCO2901/K9,VID:V06,SN:PIOS-1
 device-info       "Cisco IOS Software, C2900 Software (C2900-UNIVERSALK9-M), Version 15.4(3)M,
 hostname          Router
 ip-address        127.0.0.1
 port              10022
 name              pios1
 discovery-created true
 server            pnp1
 claimed           true
 device-added
 latest-event-type device-added
```

Serial History

The PnP Service also tracks the history of the serial numbers mapped to a device. This is particularly useful in the case where a device swap occurs and the serial number changes. As seen in the example below the serial number for device pios1 has changed from PIOS-1 to PIOS-2:

```
admin@ncs> show pnp id-state
NAME   SERIAL  SERIAL HISTORY     SERVER
-----------------------------------------
pios1  PIOS-2  [ PIOS-2 PIOS-1 ]  pnp1
```

Configuration Data

Day-0 Configuration

The configuration for a device can be set using the `day0-file` leaf in the pnp map for a device, e.g:

```
admin@ncs(config)# pnp map pios1 day0 day0-file CPE.txt
admin@ncs(config)# commit
```

The **day0-file** leaf only specifies the filename to use for a particular device and is by default set to CPE.txt. The directory to look for the file is configured in /pnp/cfg-location, e.g:

```
admin@ncs(config)# pnp cfg-location /etc/pnp/day0
admin@ncs(config)# commit
```

The configuration file is a template file and when sending the configuration to the device any template variables will be replaced with their corresponding value (or the empty string if no variable is available). Template variables come in three flavors: device specific, shared and default template variables.

Device Specific Template Variables

For each device custom template variables can be setup in the configuration for the device. These variables will only be available for the device for which they have been configured. As an example, to setup a variable **DEV_HOSTNAME** for device **pios1**:

```
admin@ncs(config)# pnp map pios1 day0 cfg-properties variable DEV_HOSTNAME value cisco.com
admin@ncs(config)# commit
```

Shared Template Variables

As well as having template variables for each device it is also possible to create a named list of shared template variables that can be used by multiple devices. Shared template variable lists are created in the `pnp` container's `day0-common` list and then referenced from the devices that want to use them:

```
admin@ncs(config)# pnp day0-common dev_vars1 variable DEV_HOSTNAME value cisco.com
admin@ncs(config)# pnp map pios day0 cfg-common dev_vars1
admin@ncs(config)# commit
```

Reserved Template Variables

The PnP package will inject some template variables, these variables cannot be overridden by any type of template varibles, see below for the list of reserved variables.

Reserved template variables

DEV_CUSTOMER_USERNAME	The username for the device as configured in the pnp map.
DEV_CUSTOMER_PASSWORD	The password for the device as configured in the pnp map.
DEV_CUSTOMER_PASSWORD_MD5	The password for the device as configured in the pnp map, hashed togheter with a random salt.
DEV_CUSTOMER_ENABLED_PASSWORD	The secondary password for the device as configured in the pnp map.

DEV_CUSTOMER_ENABLED_PASSWORD_MD5	The secondary password for the device as configured in the pnp map, hashed togheter with a random salt.
CPE_MGMT_IPV6_ADDRESS	The management IPv6 address for the device.
CPE_MGMT_IP_ADDRESS	The management IP address for the device.

Default Template Variables

By default the PnP package will inject some default template variables, see below for the list of available variables.

Default template variables

DEV_LAN_PHYSICAL_INTERFACE	The LAN interface name for the device.
DEV_WAN_PHYSICAL_INTERFACE	The WAN interface name for the device.
DAY_MINUS_ONE_FILE	The Day -1 file to restore config on device.

Config Upgrade

By default the NSO PnP server will use the PnP cli-config request to send the day-0 configuration to the device. Setting the **apply-config-upgrade** leaf to **true** for a device will make it use config-upgrade instead. This entails sending the configuration to the device via a separate HTTP GET call iniated by the PnP agent. Enabling config-upgrade for device **pios1** is done as follows:

```
admin@ncs(config)# pnp map pios1 apply-config-upgrade true
admin@ncs(config)# commit
```

Certificate Install

The NSO PnP server supports certificate install which can be enabled for a device by setting **apply-certificate-install** to **true** in the configuration for the device. In addition to this **use-ssl** for the main server needs to be set to **true** and the secondary server also needs to be enabled to serve the certificate to the client over HTTP. The following commands show how to enable certificate install for device **pios1**

```
admin@ncs(config)# pnp map pios1 apply-certificate-install true
admin@ncs(config)# pnp server use-ssl true
admin@ncs(config)# pnp secondary-server port 9090
admin@ncs(config)# commit
```

Note This section is important for SUDI.

HTTPS Client Autentication

To enable client authentication set **ca-cert-file** to the path of a valid certificate to use when authenticating the client and **verify-cpe-cert** to **true** for the PnP server:

```
admin@ncs(config)# pnp server verify-cpe-cert true
admin@ncs(config)# pnp server ca-cert-file /etc/pnp/ca-cert.pem
admin@ncs(config)# commit
```

SUDI authentication

Authourization via the PnP Client Secure Unique Device Identifier is supported. The client SUDI identifies the device as a genuine Cisco device through a built in cerfificate. Both the ACT2SUDICA and HASUDI certificates are supported.

Support for SUDI authentication can be enabled per device by setting the **sudi-authentication** leaf in the `pnp map` to **true** for the device:

```
admin@ncs(config)# pnp map pios1 sudi-authentication true
admin@ncs(config)# commit
```

NED Map

The `ned-map` list off the `pnp` container is a convenient way to set the `ned-id` and `device-type` automatically for a group of devices. It uses a regular expression to match against the `device-info` received from the PnP client to apply these values when they are not present in the `pnp/map`.

CHAPTER 4

NSO Cisco PnP Examples

Basic Example

In this example we will explain a scenario with one (1) Cisco PnP Client contacts the NSO Cisco PnP to get day-0 configuration.

Setup tracing

We need to setup a mapping between the device serial number and a configuration. But before we start we setup tracing for all devices, just to make sure we do not miss anything:

```
$ncs_cli -C -u admin
admin connected from 127.0.0.1 using console on hostmachine
admin@ncs#
Entering configuration mode terminal
admin@ncs(config)# pnp logging serial all
admin@ncs(config-serial-all)# commit
Commit complete.
```

Now it is possible to check the Erlang application log while developing. The Erlang application log is found in the `devel.log` log file on the PnP server.

PnP HELLO

The first message that is received from a PnP Client is the **PnP HELLO** request. Before this request can be handled, there needs to exist a mapping between the local and remote user.

```
admin@ncs(config)# devices authgroups group default umap system remote-name admin \
remote-password admin remote-secondary-password secret
admin@ncs(config-umap-system)# top
admin@ncs(config)# devices authgroups group default umap admin remote-name admin \
remote-password admin remote-secondary-password secret
admin@ncs(config-umap-admin)# commit
Commit complete.
```

To configure a device with serial number **1111** we need to configure this in the NSO PnP configuration:

```
admin@ncs(config)# pnp map 1111 device-name csr1 host csr1 domain cisco.com username admin \
device-typ cli ned-id cisco-ios
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# show full-configuration pnp
```

```
pnp map 1111
 host        csr1
 device-name csr1
 domain      cisco.com
 username    admin
 device-type cli
 ned-id      cisco-ios
!
```

Note In other scenarios **device-name**, **host**, **domain** etc, will have to be adjusted.

Checking the trace file located under `logs` we can see the PnP requests and responses (note that this is a simulated device).

```
From device 2016-02-08 16:39:42
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
[{pnp,[{xmlns,"urn:cisco:pnp"},
 {version,"1.0"},
 {udi,"PID:CISCO2901/K9,VID:V06,SN:1111"}],
 [{info,[{xmlns,"urn:cisco:pnp:work-info"},{correlator,"udi1"}],
 [{deviceid,[],
 [{udi,[],["PID:CISCO2901/K9,VID:V06,SN:1111"]},
 {authrequired,[],["false"]}]}]}]}]
```

The above is the Erlang representation of the data that was sent. The direction of the data is always relative the device, *From* or *To*. Below we have the xml payload:

```
<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111">
<info xmlns="urn:cisco:pnp:work-info" correlator="udi1">
<deviceid>
<udi>PID:CISCO2901/K9,VID:V06,SN:1111</udi>
<authrequired>false</authrequired></deviceid></info></pnp>
```

When the PnP Server receives this first message, it extracts the device serial number from the **udi** element. This being the first work request from the device, the PnP Server has no information about the device and asks for more information in the server response. An example of this response is shown below, *To device*. In the response the PnP Server asks the device for all **deviceInfo**:

```
To device 2016-02-08 16:39:42
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
[{pnp,[{xmlns,"urn:cisco:pnp"},
 {version,"1.0"},
 {udi,"PID:CISCO2901/K9,VID:V06,SN:1111"},
 {usr,"admin"},
 {pwd,"******"}],
 [{request,[{correlator,"udi1"},
  {version,"1.0"},
  {xmlns,"urn:cisco:pnp:device-info"}],
  [{deviceInfo,[{type,"all"}]}]}]}]

<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111" usr="admin" \
pwd="******">
<request correlator="udi1" version="1.0" xmlns="urn:cisco:pnp:device-info">
<deviceInfo type="all"></deviceInfo></request></pnp>
```

The device in turn responds with the requested data (truncated):

```
...
<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111">
<response correlator="udi1" success="1" xmlns="urn:cisco:pnp:device-info">
```

```
<udi>
<primary-chassis>PID:CISCO2901/K9,VID:V06,SN:1111</primary-chassis></udi>
<imageinfo>
<versionstring>Cisco IOS Software, C2900 Software (C2900-UNIVERSALK9-M), Version 15.4(3)M, \
RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2014 by Cisco Systems, Inc.
Compiled Mon 21-Jul-14 19:29 by prod_rel_team</versionstring>
<imagefile>flash0:c2900-universalk9-mz.SPA.154-3.M.bin</imagefile>
<imagehash></imagehash>
<returntoromreason>power-on</returntoromreason>
<bootvariable></bootvariable>
<bootldrvariable></bootldrvariable>
<configvariable></configvariable>
<configreg>0x2102</configreg>
<configregnext></configregnext></imageinfo>
<hardwareinfo>
<hostname>Router</hostname>
<vendor>Cisco</vendor>
...
```

In response to this, the PnP Server ends the communication with a *bye* message (truncated):

```
To device 2016-02-08 16:39:42
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
...
<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111" usr="admin" \
pwd="******">
<info correlator="udi1" xmlns="urn:cisco:pnp:work-info">
<workInfo>
<bye></bye></workInfo></info></pnp>
```

The PnP Server ends every PnP transaction with a *bye*' response.

PnP Client configuration

When the PnP Server and Client has made the initial handshake. The PnP Client asks the PnP Server for configuration continously using **WORK REQUEST** messages.

When the PnP Server has initial or changed configuration the PnP Server sends a **PnP Request** in a **WORK RESPONSE** that is carried in the body of a **HTTP 200 OK** response to a **WORK REQUEST**. Below the PnP Server has sent a PnP request issuing device configuration updates (in the configApply element).

```
To device 2016-02-08 16:50:14
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
...
<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111" usr="admin" \
pwd="******">
<request correlator="udi5" version="1.0" xmlns="urn:cisco:pnp:cli-config">
<configApply details="all">
<config-data>
<cli-config-data-block>no ntp
service timestamps debug datetime msec localtime show-timezone
service timestamps log datetime msec localtime show-timezone
hostname router
enable secret 5 $1$ABCD$93.Za1gPbOMwSxiQ8K2U3.
interface GigabitEthernet0/1
description Wan interface
no shutdown
aaa new-model
username admin privilege 15 secret 5 $1$ABCD$93.Za1gPbOMwSxiQ8K2U3.
```

```
aaa authentication login default none
aaa authentication login CONSOLE local
aaa authentication login VTY local
aaa authorization console
aaa authorization exec default none
aaa authorization exec CONSOLE local
aaa authorization exec VTY none
aaa authorization network default local
line con 0
 exec-timeout 30 0
 authorization exec CONSOLE
 login authentication CONSOLE
line aux 0
 no exec
line vty 0 15
 exec-timeout 30 0
 privilege level 15
 authorization exec VTY
 login authentication VTY
 transport input ssh
ip ssh version 2
no ip ssh stricthostkeycheck
ip domain lookup
ip domain name
ip name-server
ip name-server
crypto key generate rsa modulus 1024 general-keys
ipv6 unicast-routing
aaa new-model
aaa authorization network default local
vrf definition MGMT-OVERLAY
address-family ipv6
exit-address-family
crypto ikev2 authorization policy default
route set interface
crypto ikev2 redirect client
crypto ikev2 profile MGMT-OVERLAY
match identity remote fqdn domain
identity local address 127.0.0.1
authentication remote pre-share key
authentication local pre-share key
dpd 60 2 on-demand
nat keepalive 60
aaa authorization group psk list default default
crypto ipsec transform-set MGMT-ENCR esp-aes esp-sha-hmac
mode transport
crypto ipsec profile MGMT-OVERLAY
set transform-set MGMT-ENCR
set ikev2-profile MGMT-OVERLAY
interface Tunnel
vrf forwarding MGMT-OVERLAY
ipv6 address 127.0.0.1/128
tunnel source GigabitEthernet0/1
tunnel destination dynamic
tunnel protection ipsec profile MGMT-OVERLAY
crypto ikev2 client flexvpn OVERLAY
peer 1
peer 2
client connect Tunnel
ipv6 route vrf MGMT-OVERLAY  Tunnel
snmp-server source-interface traps Tunnel
ipv6 access-list SNMP-ACLv6
```

```
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
permit ipv6  any
ip access-list extended SNMP-ACLv4
 deny   ip any any
snmp-server community  RO ipv6 SNMP-ACLv6 SNMP-ACLv4
</cli-config-data-block>
</config-data>
</configApply>
</request>
</pnp>
```

In this case there are cli commands being sent. In the response from the device we can see the results from entering those commands into the device cli:

```
From device 2016-02-08 16:50:15
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
..
<pnp xmlns="urn:cisco:pnp" version="1.0" udi="PID:CISCO2901/K9,VID:V06,SN:1111">
<response xmlns="urn:cisco:pnp:cli-config" correlator="udi5" success="1">
<resultentry linenumber="1" clistring="username admin privilege 15 password 0 admin
">
<success></success></resultentry>
<resultentry linenumber="2" clistring="hostname test">
<success></success></resultentry>
<resultentry linenumber="3" clistring="ip domain-name cisco.com">
<success></success></resultentry>
<resultentry linenumber="4" clistring="crypto key generate rsa modulus 1024">
<success></success>
<text>
**CLI Line # 4: The name for the keys will be: test.cisco.com
**CLI Line # 4: % The key modulus size is 1024 bits
**CLI Line # 4: % Generating 1024 bit RSA keys, keys will be non-exportable...
**CLI Line # 4: [OK] (elapsed time was 2 seconds)
</text></resultentry>
<resultentry linenumber="5" clistring="username admin password admin">
<success></success></resultentry>
<resultentry linenumber="6" clistring="enable secret secret">
<success></success></resultentry>
<resultentry linenumber="7" clistring="ip ssh version 2">
<success></success></resultentry>
<resultentry linenumber="8" clistring="line vty 0 4">
<success></success></resultentry>
<resultentry linenumber="9" clistring="transport input ssh">
<success></success></resultentry>
<resultentry linenumber="10" clistring="login local">
<success></success></resultentry></response></pnp>
```

After this the PnP Server ends the transaction with the PnP *bye* message in a **HTTP 200 OK** response. This message is not shown.

At this stage the PnP Device has been updated with the day-0 configuration. The management IP address will be setup so that the device can be managed by NSO.

PnP Status

Under the operational data we can see the last configured ip address for management on a device:

1. **pnp-state mgmt-ip**

Unclaimed devices

Sometimes devices tries to connect to the PnP Server but there is no mapping from its' serial number and the configuration. These are *unclaimed* devices and are shown in operational data:

```
admin@ncs> show pnp unclaimed
ID
--------
1111
```

From that the id is mapped, it may take a while before a PnP request from the device is received and the device is removed from the unclaimed list.

CHAPTER 5

NSO Cisco PnP Caveats

Day-0 configuration

For the NSO PnP server to be able to configure devices after the Day-0 configuration has been applied the Day-0 configuration needs to setup the user used on the device to have privilege 15. For example by having the following line in the Day-0 configuration:

```
username $DEV_CUSTOMER_USERNAME privilege 15 secret 5 $DEV_CUSTOMER_PASSWORD_MD5
```

If AAA is used in the Day-0 for the device, HTTP AAA will also need to be enabled for the NSO PnP server to be authorized to update the the device configuration after applying the Day-0.

Alarms

Be aware that for NSO version from 4.3 and onwards alarms will be triggered when various issues occur in the NSO Cisco PnP. However due to missing support for alarms in NSO versions before 4.3 no alarms will be sent out for these versions.

CHAPTER 6

The NSO Cisco PnP Models

NSO Cisco PnP Service Model

Example 3. NSO Cisco PnP Service YANG Model

```
module tailf-ned-cisco-pnp {
  namespace "http://cisco.com/ns/nso/cfp/pnp";
  prefix pnp-service;

  import tailf-ncs {
    prefix ncs;
  }

  import tailf-common {
    prefix tailf;
  }

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import tailf-kicker { prefix kicker; }

  include tailf-ned-cisco-pnp-alarms {
    revision-date "2019-10-25";
  }

  organization "Cisco Systems";
  description "PNP Service";

  revision "2019-10-25" {
    description "Initial revision";
```

```
}

typedef sshtype {
  type enumeration {
    enum match-none {
      tailf:info "Accept any host key";
      description
        "With this setting, no SSH host key verification is done - the
         key provided by the device or cluster node may be either unknown
         or different from a 'known' key for the same key algorithm.";

    }
    enum reject-mismatch {
      tailf:info "Reject host keys that do not match the stored key";
      description
        "With this setting, the SSH host key provided by the device or
         cluster node may be unknown, but it must not be different from
         a 'known' key for the same key algorithm.";
    }
    enum reject-unknown {
      tailf:info "Reject unknown host keys";
      description
        "With this setting, the SSH host key provided by the device or
         cluster node must already be known.";
    }
  }
}

typedef device-typedef {
  type enumeration {
    enum netconf {
      tailf:code-name 'netconf_type';
    }
    enum cli {
      tailf:code-name 'cli_type';
    }
    enum generic {
      tailf:code-name 'generic_type';
    }
    enum snmp {
      tailf:code-name 'snmp_type';
    }
  }
}

typedef map-action-type {
  type enumeration {
    enum 'map-none';
    enum 'map-add';
    enum 'map-delete';
  }
}

grouping variables {
  list variable {
    key name;
    leaf name {
      type string;
      tailf:info "variable name";
    }
    choice val {
      leaf value {
```

```
        type string;
        mandatory true;
        description
          "Value of the variable";
      }
      leaf encrypted-val {
        type tailf:aes-cfb-128-encrypted-string;
        tailf:suppress-echo true;
        mandatory true;
      }
    }
  }
}

grouping config-restore {
  choice config-restore {
    leaf config-restore-file {
      type string;
      default "flash:day--1-config";
    }
    leaf config-restore-uri {
      type string;
    }
    leaf config-restore-disabled {
      type empty;
    }
    leaf config-erase {
      type empty;
    }
  }
}

grouping device-state {
  leaf serial {
    type string;
  }

  leaf udi {
    type string;
  }

  leaf username {
    type string;
  }

  leaf password {
    type tailf:aes-cfb-128-encrypted-string;
    tailf:suppress-echo true;
  }

  leaf device-info {
    type string;
  }

  leaf hostname {
    type string;
  }

  leaf ip-address {
    type inet:ip-address;
  }
```

```
leaf mgmt-ip {
  type inet:ip-address;
  description
    "Management address configured on the device.";
}

leaf port {
  type inet:port-number;
}

leaf name {
  type string;
}

leaf wan-interface {
  type string;
}

leaf lan-interface {
  type string;
}

leaf configured {
  type boolean;
}

leaf-list configuration-file {
  type string;
  ordered-by user;
}

leaf certificate-installed {
  type boolean;
}

leaf request {
  type enumeration {
    enum 'device-info' {
      tailf:code-name "device-info-x";
    }
    enum 'config';
    enum 'none';
    enum 'backoff';
    enum 'config-upgrade';
    enum 'certificate-install';
    enum 'sudi-auth';
  }
}

leaf is-netsim {
  type boolean;
}

leaf need-clean {
  type boolean;
}

leaf pending-exec {
  type string;
}

leaf sudi-authenticated {
```

```
  type boolean;
  description
    "Indicates whether the device has been authenticated using SUDI.";
}

leaf sudi-sid {
   type tailf:aes-cfb-128-encrypted-string;
   tailf:suppress-echo true;
   description
     "SUDI session ID.";
}

leaf last-error {
  type string;
}

leaf last-contact {
  type string;
  tailf:cli-preformatted;
}

leaf last-clean {
  type uint32;
}

leaf snmp {
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
  description
      "Snmp community name or authgroup depending on the snmp version used.";
}

leaf discovery-created {
  description
    "If the client sends device info with discovery-created equal to true,
     the pnp-server sets this value to true, to indicate that the
     device used the global pnp-server to find this pnp-server.
     If the device omits sending the discovery-created attribute,
     or if it is set to false, pnp-server sets this value to false.";
  type boolean;
}

uses config-restore {
  refine config-restore/config-restore-file {
    description
      "Name of file on CPE that will be used when restoring device to
       initial config. If not specified the value set in the
       interface-map or a default one will be used.";
  }
  refine config-restore/config-restore-uri {
    description
      "URI of file that will be used when restoring device to
       initial config. If not specified the value set in the
       interface-map or a default one will be used.";

  }
  refine config-restore/config-restore-disabled {
    description
      "Disable config restore.";
  }
}
```

```
    leaf server {
      type string;
    }

    leaf queued {
      type boolean;
    }
    leaf claimed {
      type boolean;
    }
    leaf device-added {
      type empty;
    }

    leaf latest-event-type {
      type string;
    }
  }

  container pnp {
    container internal {
      container actions {
        tailf:action create-device {
          tailf:actionpoint pnp-create-device;
          input {
            uses ncs:post-action-input-params;
          }
          output {}
        }
        tailf:action fetch-host-keys {
          tailf:actionpoint pnp-fetch-host-keys;
          input {
            uses ncs:post-action-input-params;
          }
          output {}
        }
        tailf:action sync-from {
          tailf:actionpoint pnp-sync;
          input {
            uses ncs:post-action-input-params;
          }
          output {}
        }
      }
    }
    tailf:action handle-netconf-notifications {
      tailf:hidden full;
      tailf:actionpoint pnp-netconf-notifications;
      input {
        uses kicker:action-input-params;
      }
      output {
      }
    }
    tailf:action migration {
      tailf:actionpoint pnp_migration;
      tailf:info "Migration of services from PNP 2 to PNP 3";
      input {
        leaf confd-server {
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
```

```
      tailf:info "Name of ConfD device hosting PNP server";
      must "starts-with(deref(.)/../ncs:device-type/ncs:netconf/ncs:ned-id, 'confd-pnp-ne
        tailf:dependency ".";
        error-message "Device has to be of confd-pnp-ned ned-type to be used as a PnP ser
      }
      mandatory true;
    }
  }
  output {
    leaf result {
      type string;
    }
    leaf success {
      type boolean;
    }
  }
}
tailf:action exec {
  tailf:actionpoint pnp_exec;
  input {
    choice device {
      leaf all {
        type empty;
        tailf:info "'all' will execute the command on all PnP devices";
      }
      leaf serial {
        type leafref {
          path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
        }
        tailf:info "Serial of the device to execute command on";
      }
      leaf server {
        type leafref {
          path "/pnp-service:pnp/pnp-service:servers/pnp-service:name";
        }
        tailf:info "All PnP devices on this Server to execute command on";
      }
      mandatory true;
    }
    leaf command {
      type string {
        tailf:info "Native command to send to device.";
      }
      mandatory true;
    }
  }
  output {
    leaf result {
      type string;
      description "Status of action";
    }
  }
  tailf:info "Execute command on PNP device(s).";
}

tailf:action reset {
  tailf:actionpoint pnp_reset;
  input {
    choice device {
      leaf all {
        type empty;
        tailf:info "'all' will reset all PNP devices";
```

```
      }
      leaf serial {
        type leafref {
          path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
        }
        tailf:info "Serial of the device to reset";
      }
      leaf server {
        type leafref {
          path "/pnp-service:pnp/pnp-service:servers/pnp-service:name";
        }
        tailf:info "All PnP devices on this Server to reset";
      }
      mandatory true;
    }
    leaf clean {
      tailf:info "Restore device to day--1 config.";
      type empty;
    }
    leaf username {
      tailf:info "Explicitly set username. This setting overrides pnp-map
                  for the reset.";
      type string;
    }
    leaf password {
      tailf:info "Explicitly set password. This setting overrides pnp-map
                  for the reset.";
      type string;
      tailf:suppress-echo true;
    }
  }
  output {
    leaf result {
      type string;
      description "Status of action";
    }
  }
  tailf:info "Restore PNP device(s) to day--1 configuration.";
}

tailf:action delete {
  tailf:actionpoint pnp_delete;
  input {
    choice device {
      leaf all {
        type empty;
        tailf:info "'all' will delete all PnP devices";
      }
      leaf serial {
        type leafref {
          path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
        }
        tailf:info "Serial of the device to delete";
      }
      leaf server {
        type leafref {
          path "/pnp-service:pnp/pnp-service:servers/pnp-service:name";
        }
        tailf:info "All PnP devices on this Server to delete";
      }
      mandatory true;
    }
```

```
  }
  output {
    leaf result {
      type string;
      description "Status of action";
    }
  }
  tailf:info "Delete PNP device(s).";
}

tailf:action trace {
  tailf:actionpoint pnp_trace;
  input {
    choice enable-choice {
      leaf enable {
        type empty;
        tailf:info "Enable tracing of PNP communication";
      }
      leaf disable {
        type empty;
        tailf:info "Disable tracing of PNP communication";
      }
    }

    leaf "show-xml" {
      type empty {
        tailf:info "This option shows the tracing inline in the CLI. This
                    option only has effect together with 'enable'.
                    Trace logs will otherwise be stored under
                    '../pnp:logging/directory";
      }
    }
  }
  output {
    leaf result {
      type string;
      description "Status of action";
    }
  }
  tailf:info "Setup PNP traceing for PNP device(s).";
}

tailf:action device-state {
  tailf:actionpoint pnp_device_state;
  input {
    leaf serial {
      type leafref {
        path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
      }
      mandatory true;
      tailf:info "Serial of the device to get the state for";
    }
    leaf nso-key {
      type string;
      tailf:suppress-echo true;
      tailf:info "The key corresponding to the one setup for the device";
    }
  }
  output {
    uses device-state;
    leaf result {
      type string;
```

```
      description "Status of action";
    }
  }
  tailf:info "Get device state.";
}

tailf:action redeploy-pending {
  tailf:actionpoint pnp_redeploy_pending;
  tailf:hidden debug;
  input {
    leaf serial {
      type leafref {
        path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
      }
      when "/pnp-service:pnp/pnp-service:state/pnp-service:pending-exec";
      mandatory true;
      tailf:info "Serial of the device to redeploy pending action";
    }
  }
  output {
  }
  tailf:info "Redeploy pending.";
}

tailf:action delete-pending {
  tailf:actionpoint pnp_delete_pending;
  tailf:hidden debug;
  input {
    leaf serial {
      type leafref {
        path "/pnp-service:pnp/pnp-service:state/pnp-service:serial";
      }
      when "/pnp-service:pnp/pnp-service:state/pnp-service:pending-exec";
      mandatory true;
      tailf:info "Serial of the pending device to delete";
    }
  }
  output {
  }
  tailf:info "Delete Pending Map of this Serial.";
}

list state {
  config false;
  tailf:cdb-oper {
    tailf:persistent true;
  }
  key serial;
  uses device-state;
}

list id-state {
  config false;
  tailf:cdb-oper {
    tailf:persistent true;
  }
  key name;
  leaf name {
    type string;
  }

  leaf serial {
```

```
    type string {
      tailf:info "WORD;;Serial number of device";
      length "1..32";
    }
  }

  leaf-list serial-history {
    ordered-by user;
    type string {
      tailf:info "WORD;;Serial number of device";
      length "1..32";
    }
  }

  leaf server {
    type string {
    }
  }
}

list map-plan {
  config false;
  tailf:cdb-oper {
    tailf:persistent true;
  }
  key "id";
  leaf id {
    type string;
  }
  uses ncs:nano-plan-data;
}

list map {
  uses ncs:service-data;
  ncs:servicepoint cisco-pnp-map;
  tailf:info "Mapping between serial number and device.";

  key id;
  unique serial;

  description
    "Map from device serial number to device name. This
     information is used when mounting the device into the
     NCS device tree.";

  leaf id {
    type string {
      tailf:info "WORD;;pnp map id";
    }
  }

  leaf serial {
    type string {
      length "1..32";
    }
    tailf:info "Serial number of device";
    mandatory true;
    description
      "Serial number of device. The device sends this as
       part of the URI.";
  }
```

```
choice device-name-choice {
  leaf device-name {
    type string;
    tailf:info "Name to mount the device under in NCS";
    mandatory true;
    description
      "This is the name under which the device is mounted in
       NCS, ie in the /devices/device tree.";
  }
  leaf use-hostname {
    type enumeration {
      enum hostname;
      enum serial-with-hostname;
    }
    tailf:info "Name from device PNP protocol to mount the device under in NCS";
    mandatory true;
    description
      "This is the name under which the device is mounted in
       NCS, ie in the /devices/device tree.";
  }
}

leaf apply-certificate-install {
  type boolean;
  tailf:info "Do certificate installation on the client when connecting
     on the secondary server.";
  description
    "Do certificate installation on the client when connecting
     on the secondary server.";
}

leaf authgroup {
  type leafref {
    path "/ncs:devices/ncs:authgroups/ncs:group/ncs:name";
  }
  tailf:info "Credentials to use when managing the device. If no authgroup is
     specified a new authgroup will be created with username, password
     and secondary password taken from the PnP device config";
  description
    "Credentials to use when managing the device. If no authgroup is
     specified a new authgroup will be created with username, password
     and secondary password taken from the PnP device config.
     NOTE: If an authgroup is specified make sure that the config
     creates the auth config configured in the authgroup on the device
     with the Day0 config.";
}

leaf username {
  type string;
  tailf:info "Username for configuring device";
  mandatory true;
  description
    "The username that will be used when communicating using PnP
     with the device and for managing a device if no authgrup has
     been specified.";
}

leaf password {
  tailf:info
    "The password that will be used when communicating using PnP
     with the device and managing the device.";
```

```
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
}

leaf sec-password {
  tailf:info "Define the second password for the CPE.";
  type tailf:aes-cfb-128-encrypted-string;

  tailf:suppress-echo true;
}

leaf device-type {
  type device-typedef;
  tailf:info "The device-type defines how the communication towards
              the device is carried out.";
}

leaf ned-id {
  type string;
  tailf:info "NED id to use when mounting device.";
  description
    "NED id to use when mounting device in the devices tree in NCS.";
}

leaf port {
  type inet:port-number;
  tailf:info "Management port for the device";
  mandatory true;
  description
    "Managment port for the device";
}

container day0 {
  presence "The device will have day0 configuration data applied";
  leaf-list day0-file {
    type string;
    tailf:info "Day0 config filename.";
    description
      "Files to use for Day0 config for the device.";
    ordered-by user;
  }

  container cfg-properties {
    uses variables;
    tailf:info "Device specific template variables.";
  }

  leaf cfg-common {
    type leafref {
      path "../../../day0-common/cfgName";
    }
  }
}

leaf apply-config-upgrade {
  type boolean;
  tailf:info "Set to true if pnp server needs to send configuration to
     device through config-upgrade";
  description
    "Set to true if pnp server need to send configuration to
     device through config-upgrade ";
}
```

```
choice mgmt-ip {
  leaf mgmt-ip-address {
    type inet:ip-address;
    tailf:info "management ip address of the device.";
  }

  leaf auto-update-mgmt-ip {
    type empty;
    tailf:info "If source PNP client source IP changes, the source IP
        will be used to update the management IP address.";
    description
      "If auto-update-mgmt-ip is set and the PNP client source IP address
     is changed (for example through DHCP), the source IP address will be
     used to update the management IP address in the NSO device tree.";
  }
}

// TODO: What is this?
leaf commit-queue-temp {
  type boolean;
  default false;
  tailf:info "Use commit-queue for the device if true.";
  description
    "Use commit-queue for the device if true.";
}

leaf sudi-authentication {
  type boolean;
  default false;
  tailf:info "Set to true if SUDI authentication is needed for the device";
  description
    "Set to true if SUDI authentication is needed for the device";
}

leaf nso-key {
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
  tailf:info "Key that needs to be specified when requesting the device state
     using the device-state action.";
  description
    "Key that needs to be specified when requesting the device state
     using the device-state action. If not set allow the action
     regardless of the key supplied when executing the action.";
}

leaf managed {
  type boolean;
  default false;
  tailf:info "The device will be added to CDB device tree";
  description
    "The device will be added to CDB device tree
     if true.";
}

leaf use-relative-url {
  type boolean;
  default false;
  tailf:info "Relativce URL should be used in config upgrade and
      certificate install work requests.";
  description
    "Set to true if a relative URL should be used in config upgrade
```

```
        and certificate install work requests";
    }

    leaf need-clean {
      type empty;
      tailf:info "If set, send Day--1 config to the device when it is deleted from the
         PnP map.";
      description
        "If set, send Day--1 config to the device when it is deleted from the
         PnP map";
    }

    uses config-restore {
      refine config-restore/config-restore-file {
        tailf:info "Name of file on CPE that will be used when restoring device to
           initial config.";
        description
          "Name of file on CPE that will be used when restoring device to
           initial config.";
      }
      refine config-restore/config-restore-uri {
        tailf:info "URI of file that will be used when restoring device to
           initial config.";
        description
          "URI of file that will be used when restoring device to
           initial config.";
      }
      refine config-restore/config-restore-disabled {
        tailf:info "Disable config restore.";
        description
          "Disable config restore.";
      }
    }
  }

  list day0-common {
    key cfgName;
    leaf cfgName {
      type string;
      description
        "Name of the common variables to be shared for pnp devices";
    }
    uses variables;
  }

  container state-value {
    leaf device-sync-retry-attempts {
      tailf:info "Number of attempts to sync the device if sync fails the
                  first time after device is successfully added and
                  configured.";
      type uint8 {
        tailf:info "<INT>;; Retry attempts";
      }
      default 3;
    }
  }

  list ned-map {
    key expr;

    leaf expr {
      type string;
```

```
    description
      "Regular expression to match agains device-info versionstring.";
  }

  leaf device-type {
    type device-typedef;
    mandatory true;
    tailf:info "The device-type defines how the communication towards
                the device is carried out.";
    description "Please refer to the user guide for more information
                 about the difference between different device types.";
  }

  leaf ned-id {
    type string {
      tailf:info "WORD;;NED id to use when mounting device.";
    }
    mandatory true;
    description
      "NED id to use when mounting device in the devices tree
       in NCS.";
  }
}

container logging {
  list serial {
    key name;
    leaf name {
      type string {
        tailf:info "UDI or all;;Name of UDI to log, 'all' will apply
                    to all CPEs.";
      }
    }
  }
  tailf:info "PNP Server logging configuration.";
}

leaf host-key-verification-type {
  type enumeration {
    enum none {
      tailf:code-name "glob_match_none";
      tailf:info "Accept any host key";
      description
      "With this setting, no SSH host key verification is done - the
       key provided by the device or cluster node may be either unknown
       or different from a 'known' key for the same key algorithm.";

    }
    enum reject-mismatch {
      tailf:info "Reject host keys that do not match the stored key";
      description
        "With this setting, the SSH host key provided by the device or
         cluster node may be unknown, but it must not be different from
         a 'known' key for the same key algorithm.";
    }
    enum reject-unknown {
      tailf:info "Reject unknown host keys";
      description
        "With this setting, the SSH host key provided by the device or
         cluster node must already be known.";
    }
  }
```

```
  default none;
  tailf:info "SSH key verification.";
}

leaf cfg-location {
  description "The location of the day0 files";
  type string;
}

list servers {
  uses ncs:service-data;
  ncs:servicepoint "pnp-server-config-servicepoint";

  key name;
  leaf name {
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
    must "starts-with(deref(.)/../ncs:device-type/ncs:netconf/ncs:ned-id, 'confd-pnp-ned'
      tailf:dependency ".";
      error-message "Device has to be of confd-pnp-ned ned-type to be used as a PnP serve
    }
  }

  container server {
    presence "Configure the PnP server";
    tailf:info "PNP Server configuration.";

    leaf ip-address {
      type inet:ip-address {
        tailf:info "PNP Server ip address.";
      }
      default "0.0.0.0";
      description
        "The address on which the PnP server will listen
         for incoming requests. Default is to listen on
         all addresses available.";
    }

    leaf-list additional-ip-address {
      type inet:ip-address {
        tailf:info "Additional PnP Server IP address.";
      }
      description
        "Additional IP address on which the PnP server will listen
         for incoming requests.";
    }

    leaf port {
      type inet:port-number {
        tailf:info "PNP Server port.";
      }
      default 9191;
      description
        "The port on which the PnP server will listen
         for incoming requests.";
    }

    leaf use-ssl {
      type boolean {
        tailf:info "Enable or disable HTTPS.";
      }
```

```
      default true;
      description
        "If set to true, the PnP server will use the https
         protocol. If false, http will be used.";
    }

    leaf verify-cpe-cert {
      type boolean;
      default false;
      description
        "If set to true, the PnP server will do require client
         authentication. If false, no client authentication will
         be required.";
    }

    leaf ca-cert-file {
      type string;
      default "ca_cert";
      description
        "The CA certificate to use when doing client authentication";
    }
  }

  container proxy-servers {
    presence "Configure the PnP proxy server";
    tailf:info "Proxy Server Support.";

    description
      "If proxy servers are used in front of the PNP server the
       reported IP address of the device (as stored in
       /pnp-state/device/ip-address) will be the proxy IP address
       instead of the device IP address.

       If the proxy server supports forwarding the client IP address
       using the X-Forwarded-For header, the PNP server can be
       configured to use that as the client IP address.

       By default the PNP server will not honor the X-Forwarded-For
       header, it must be enabled here, either by listing the IP
       addresses of the proxies (in which case X-Forwarded-For will
       be honored only from those IP addresses) or by setting
       'allow-any' to 'true'.";

    choice x-forwarded-for {
      leaf allow-any {
        type empty;
        description
          "If set to true, then client IP address will be taken from
           the X-Forwarded-For header (if present).";
        tailf:info "Always take client IP from X-Forwarded-For header";
      }
      leaf-list allow-from {
        type inet:ip-address;
        description "Only honor X-Forwarded-For from these addresses.";
        tailf:info "Proxy Server addresses";
      }
    }
  }

  container secondary-server {
    presence "Configure the PnP secondary server";
    tailf:info "Secondary PnP Server configuration used for certificate
```

```
                installation.";

    leaf ip-address {
      type inet:ip-address {
        tailf:info "PnP secondary server ip address.";
      }
      default "0.0.0.0";
      description
        "The address on which the secondary PnP server will listen
         for incoming requests. Default is to listen on
         all addresses available.";
    }

    leaf-list additional-ip-address {
      type inet:ip-address {
        tailf:info "Additional secondary PnP server IP address.";
      }
      description
        "Additional IP address on which the secondary PnP server will listen
         for incoming requests.";
    }

    leaf port {
      type inet:port-number {
        tailf:info "PnP secondary server port.";
      }
      default 9090;
      description
        "The port on which the secondary PnP server will listen
         for incoming requests.";
    }
  }

  container state-value {
    presence "Configure PnP server state value";
    leaf secure-pnp {
      description
        "Enable secure PnP which will make the PnP server never send out
         usr/pwd to a device before the device has completed SUDI
         authentication. For this setting to work one has to enable SUDI
         for all devices otherwise most PnP work requests will fail since
         the PnP server won't send out the username and password required
         to pass AAA on devices.";
      type boolean;
      default true;
    }

    leaf backoff-timeout {
      tailf:info "Timeout between CPE re-sync with the PnP. This parameter
                  controls how long it will take for the PnP server to
                  notice that a device has changed ip address.";
      type uint32 {
        tailf:info "<INT>;;Timeout in seconds between client reconnect.";
      }
      default "210";
    }

    leaf terminate-when-done {
      type empty;
      description
        "If present PnP process will be terminated and CPE will no longer
         re-sync with PnP server after the device onboard.";
```

```
    }
  }

  list device-map {
    key expr;

    leaf expr {
      type string;
      tailf:info "Reg expr used to match against device-info versionstring.";
      description
        "Regular expression to match agains device-info versionstring.";
    }

    leaf wan {
      type string;
      tailf:info "WAN interface on CPE device.";
      description
        "WAN interface on CPE device.";
    }

    leaf lan {
      type string;
      tailf:info "LAN interface on CPE device.";
      description
        "LAN interface on CPE device.";
    }

    uses config-restore {
      refine config-restore {
        default config-restore-file;
      }
      refine config-restore/config-restore-file {
        tailf:info
          "Name of file on cpe that will be used when restoring device to
           initial config. The pnp server will send the commands: \n
           copy <config-restore-file> startup-config \n reload";
      }
      refine config-restore/config-restore-uri {
        tailf:info
          "URI of file that will be used when restoring device to
           initial config. The pnp server will send the commands: \n
           copy <config-restore-uri> startup-config \n reload";
      }
      refine config-restore/config-restore-disabled {
        description
          "Disable config restore.";
      }
    }

    leaf trustpoint-label {
      tailf:info "Label to use for the trustpoint in certificate install.";
      description
        "Label to use for the trustpoint in certificate install.";
      type union {
        type string;
        type enumeration {
          enum random;
        }
      }
    }

    leaf server-side-certificate {
```

```
        type string;
        tailf:info
          "Certificate (from file) to be sent to the PnP agent in the certificate
           install request.";
        description
          "Certificate to be sent to the PnP agent in the certificate
           install request. This is not mandatory, if this leaf is not
           set, the Server host certificate will be sent.";
      }

      leaf disable-serial-check {
        type empty;
        tailf:info "If present will disable the SUDI Serial check.";
        description "If present will disable the SUDI Serial check.  This will
        prevent matching the SUDI with the serial, however the SUDI key will still
        be verified.";
      }
    }

    container logging {
      presence "Configure PnP server logging";

      list serial {
        key name;
        leaf name {
          type string {
            tailf:info "UDI or all;;Name of UDI to log, 'all' will apply
                        to all CPEs.";
          }
        }
      }
      tailf:info "PNP Server logging configuration.";
    }

    leaf wait-after-reload-time {
      tailf:info "Wait time between issuing reload and applying day0
                  configuration for CPE";
      type uint32 {
        tailf:info "<INT>;;Wait time in seconds between reloading and
                    applying day0 for CPE configuration.";
      }
    }
  }
 }
}
```

NSO Cisco PnP Service Nano-Service Model

Example 4. NSO Cisco PnP Service Nano-Service YANG Model

```
module tailf-ned-nano-services-pnp {
  namespace "http://cisco.com/ns/nso/cfp/tailf-ned-nano-services-pnp";
  prefix ns-pnp;

  import tailf-common { prefix tailf; }
  import tailf-ncs { prefix ncs; }
  import tailf-ned-cisco-pnp { prefix pnp-service; }

  identity managed-device {
    base ncs:plan-component-type;
  }
```

```
identity unmanaged-device {
  base ncs:plan-component-type;
}

identity sync {
  base ncs:plan-state;
}

identity device-created {
  base ncs:plan-state;
}

identity device-connected {
  base ncs:plan-state;
}

identity fetch-host-keys {
  base ncs:plan-state;
}

ncs:plan-outline cisco-pnp-map-plan {
  ncs:component-type "ncs:self" {
    ncs:state "ncs:init" {
      ncs:create {
        ncs:nano-callback;
      }
    }
    ncs:state "ns-pnp:device-connected" {
      ncs:create {
        ncs:pre-condition {
          ncs:monitor "/pnp-service:pnp/pnp-service:id-state[name=$ID]" {
            // TODO: Is this always the case, i.e it's enough that we
            // see the device to send down the map config?
            ncs:trigger-expr "server";
          }
        }
        ncs:nano-callback;
      }
    }

    ncs:state "ncs:ready" {
      ncs:create {
        ncs:pre-condition {
          ncs:monitor "$PLAN" {
            // All other components have reached ready or have
            // failed
            ncs:trigger-expr
              "not(component[type != 'ncs:self']/state[name = 'ncs:ready'][status != 'reached'
            + "or boolean(failed)";
          }
        }
        ncs:nano-callback;
      }
    }
  }

  ncs:component-type "ns-pnp:managed-device" {
    ncs:state "ncs:init";

    ncs:state "ns-pnp:device-created" {
      ncs:create {
```

```
      ncs:pre-condition {
        ncs:monitor "/pnp-service:pnp/pnp-service:state[serial=$SERIAL]" {
          // TODO: Is this always the case, i.e it's enough that we
          // see the device to send down the map config?
          ncs:trigger-expr "device-added";
        }
      }
      ncs:post-action-node "/pnp-service:pnp/pnp-service:internal/pnp-service:actions" {
        ncs:action-name "create-device";
      }
    }
  }
  ncs:state "ns-pnp:fetch-host-keys" {
    ncs:create {
      ncs:pre-condition {
        ncs:monitor "$PLAN/component[type='ns-pnp:managed-device'][name=$ID]/state[name='
          ncs:trigger-expr "post-action-status = 'create-reached'";
        }
      }
      ncs:post-action-node "/pnp-service:pnp/pnp-service:internal/pnp-service:actions" {
        ncs:action-name "fetch-host-keys";
      }
      // ncs:post-action-node "/ncs:devices/ncs:device[ncs:name=$SERIAL]/ncs:ssh" {
      //   ncs:action-name "fetch-host-keys";
      // }
    }
  }
  ncs:state "ns-pnp:sync" {
    ncs:create {
      ncs:pre-condition {
        ncs:monitor "$PLAN/component[type='ns-pnp:managed-device'][name=$ID]/state[name='
          ncs:trigger-expr "post-action-status = 'create-reached'";
        }
      }
      ncs:post-action-node "/pnp-service:pnp/pnp-service:internal/pnp-service:actions" {
        ncs:action-name "sync-from";
      }
      // ncs:post-action-node "/ncs:devices/ncs:device[ncs:name=$SERIAL]" {
      //   ncs:action-name "sync-from";
      // }
    }
  }
  ncs:state "ncs:ready" {
    ncs:create {
      ncs:pre-condition {
        ncs:monitor "$PLAN/component[type='ns-pnp:managed-device'][name=$ID]/state[name='
          ncs:trigger-expr "post-action-status = 'create-reached'";
        }
      }
    }
  }
}
ncs:component-type "ns-pnp:unmanaged-device" {
  ncs:state "ncs:init";
  ncs:state "ncs:ready" {
    ncs:create {
      ncs:pre-condition {
        ncs:monitor "/pnp-service:pnp/pnp-service:state[serial=$SERIAL]" {
          // TODO: Is this always the case, i.e it's enough that we
          // see the device to send down the map config?
          ncs:trigger-expr "device-added";
        }
```

```
        }
      }
    }
  }
}

ncs:service-behavior-tree cisco-pnp-map {
  ncs:plan-outline-ref "ns-pnp:cisco-pnp-map-plan";
  ncs:plan-location "/pnp-service:pnp/pnp-service:map-plan";

  ncs:selector {
    ncs:selector {
      ncs:variable "ID" {
        ncs:value-expr "id";
      }
      ncs:variable "SERIAL" {
        ncs:value-expr "serial";
      }
      ncs:create-component "'self'" {
        ncs:component-type-ref "ncs:self";
      }

      ncs:create-component "$ID" {
        ncs:pre-condition {
          ncs:monitor "$SERVICE/managed" {
            ncs:trigger-expr ". = 'true'";
          }
        }
        ncs:component-type-ref "ns-pnp:managed-device";
      }

      ncs:create-component "$ID" {
        ncs:pre-condition {
          ncs:monitor "$SERVICE/managed" {
            ncs:trigger-expr ". = 'false'";
          }
        }
        ncs:component-type-ref "ns-pnp:unmanaged-device";
      }
    }
  }
}
```

NSO Cisco PnP Service Alarms Model

Example 5. NSO Cisco PnP Service alarms YANG Model

```
submodule tailf-ned-cisco-pnp-alarms {
  belongs-to tailf-ned-cisco-pnp {
    prefix pnp-service;
  }

  import tailf-ncs-alarms {
    prefix al;
  }

  organization "Cisco";
  description "PNP Service";

  revision "2019-10-25" {
```

```
    description "Initial revision";
  }

  identity pnp-alarm {
    base al:alarm-type;
    description "Alarms raised by the tailf-ned-cisco-pnp package.";
  }

  identity open-log-failure {
    base pnp-alarm;
    description
      "Failed to open the logfile for writing.";
  }

  identity config-apply-error {
    base pnp-alarm;
    description
      "Error trying to serve config apply request";
  }

}
```

NSO Cisco PnP Server Model

Example 6. NSO Cisco PnP Server YANG Model

```
module cisco-confd-pnp {
  namespace "http://cisco.com/ns/nso/cfp/pnp-server";
  prefix confd-pnp;

  import tailf-common {
    prefix tailf;
  }

  import ietf-inet-types {
    prefix inet;
  }

  organization "Cisco";
  description "CONFD PNP Server";

  revision 2019-10-25 {
    description "Initial revision";
  }

  grouping variables {
    list variable {
      key name;
      leaf name {
        type string;
        tailf:info "variable name";
      }
      choice val {
        leaf value {
          type string;
          mandatory true;
          description
            "Value of the variable";
        }
        leaf encrypted-val {
          type tailf:aes-cfb-128-encrypted-string;
```

```
        tailf:suppress-echo true;
        mandatory true;
      }
    }
  }
}

grouping config-restore {
  choice config-restore {
    leaf config-restore-file {
      type string;
      default "flash:day--1-config";
    }
    leaf config-restore-uri {
      type string;
    }
    leaf config-restore-disabled {
      type empty;
    }
    leaf config-erase {
      type empty;
    }
  }
}

grouping device-state {
  leaf serial {
    type string;
  }

  leaf udi {
    type string;
    default "";
  }

  leaf username {
    type string;
  }

  leaf password {
    type tailf:aes-cfb-128-encrypted-string;
    tailf:suppress-echo true;
  }

  leaf device-info {
    type string;
  }

  leaf hostname {
    type string;
  }

  leaf ip-address {
    type inet:ip-address;
    default "0.0.0.0";
  }

  leaf mgmt-ip {
    type inet:ip-address;
    default "0.0.0.0";
    description
      "Management address configured on the device.";
```

```
}

leaf port {
  type inet:port-number;
  default 0;
}

leaf wan-interface {
  type string;
}

leaf lan-interface {
  type string;
}

leaf configured {
  type boolean;
  default false;
}

leaf-list configuration-file {
  type string;
  ordered-by user;
}

leaf certificate-installed {
  type boolean;
  default false;
}

leaf request {
  type enumeration {
    enum 'device-info' {
      tailf:code-name "device-info-x";
    }
    enum 'config';
    enum 'none';
    enum 'backoff';
    enum 'config-upgrade';
    enum 'certificate-install';
    enum 'sudi-auth';
    enum 'reload';
  }
  default none;
}

leaf provisioned {
  type boolean;
  default false;
}

leaf added {
  type boolean;
  default false;
}

leaf is-netsim {
  type boolean;
  default false;
}

leaf need-clean {
```

```
  type boolean;
  default false;
}

leaf pending-exec {
  type string;
}

leaf sudi-authenticated {
  type boolean;
  default false;
  description
    "Indicates whether the device has been authenticated using SUDI.";
}

leaf sudi-sid {
   type tailf:aes-cfb-128-encrypted-string;
   tailf:suppress-echo true;
   description
     "SUDI session ID.";
}

leaf last-error {
  type string;
}

leaf last-contact {
  type string;
  tailf:cli-preformatted;
}

leaf last-clean {
  type uint32;
  default 0;
}

leaf discovery-created {
  description
    "If the client sends device info with discovery-created equal to true,
     the pnp-server sets this value to true, to indicate that the
     device used the global pnp-server to find this pnp-server.
     If the device omits sending the discovery-created attribute,
     or if it is set to false, pnp-server sets this value to false.";
  type boolean;
  default false;
}

uses config-restore {
  refine config-restore/config-restore-file {
    description
      "Name of file on CPE that will be used when restoring device to
       initial config. If not specified the value set in the
       interface-map or a default one will be used.";
  }
  refine config-restore/config-restore-uri {
    description
      "URI of file that will be used when restoring device to
       initial config. If not specified the value set in the
       interface-map or a default one will be used.";

  }
  refine config-restore/config-restore-disabled {
```

```
      description
        "Disable config restore.";
    }
  }
}

container pnp {
  tailf:action exec {
    tailf:actionpoint pnp_exec;
    input {
      choice device {
        leaf all {
          type empty;
          tailf:info "'all' will execute the command on all PnP devices";
        }
        leaf serial {
          type leafref {
            path "/confd-pnp:pnp-state/confd-pnp:device/confd-pnp:serial";
          }
          tailf:info "Serial of the device to execute command on";
        }
        mandatory true;
      }
      leaf command {
        type string {
          tailf:info "Native command to send to device.";
        }
        mandatory true;
      }
    }
    output {
    }
    tailf:info "Execute command on PNP device(s).";
  }

  tailf:action reset {
    tailf:actionpoint pnp_reset;
    input {
      choice device {
        leaf all {
          type empty;
          tailf:info "'all' will reset all PNP devices";
        }
        leaf serial {
          type leafref {
            path "/confd-pnp:pnp-state/confd-pnp:device/confd-pnp:serial";
          }
          tailf:info "Serial of the device to reset";
        }
        mandatory true;
      }
      leaf clean {
        tailf:info "Restore device to day--1 config.";
        type empty;
      }
      leaf username {
        tailf:info "Explicitly set username. This setting overrides pnp-map
                    for the reset.";
        type string;
      }
      leaf password {
        tailf:info "Explicitly set password. This setting overrides pnp-map
```

```
                    for the reset.";
        type string;
        tailf:suppress-echo true;
      }
    }
    output {
    }
    tailf:info "Restore PNP device(s) to day--1 configuration.";
  }

  tailf:action delete {
    tailf:actionpoint pnp_delete;
    input {
      choice device {
        leaf all {
          type empty;
          tailf:info "'all' will delete all PnP devices";
        }
        leaf serial {
          type leafref {
            path "/confd-pnp:pnp-state/confd-pnp:device/confd-pnp:serial";
          }
          tailf:info "Serial of the device to delete";
        }
        mandatory true;
      }
    }
    output {
    }
    tailf:info "Delete PNP device(s).";
  }

  tailf:action trace {
    tailf:actionpoint pnp_trace;
    input {
      choice enable-choice {
        leaf enable {
          type empty;
          tailf:info "Enable tracing of PNP communication";
        }
        leaf disable {
          type empty;
          tailf:info "Disable tracing of PNP communication";
        }
      }

      leaf "show-xml" {
        type empty {
          tailf:info "This option shows the tracing inline in the CLI. This
                      option only has effect together with 'enable'.
                      Trace logs will otherwise be stored under
                      '../confd-pnp:logging/directory";
        }
      }
    }
    output {
    }
    tailf:info "Setup PNP traceing for PNP device(s).";
  }

  tailf:action device-state {
    tailf:actionpoint pnp_device_state;
```

```
  input {
    leaf serial {
      type leafref {
        path "/confd-pnp:pnp-state/confd-pnp:device/confd-pnp:serial";
      }
      mandatory true;
      tailf:info "Serial of the device to get the state for";
    }
    leaf nso-key {
      type string;
      tailf:suppress-echo true;
      tailf:info "The key corresponding to the one setup for the device";
    }
  }
  output {
    uses device-state;
  }
  tailf:info "Get device state";
}

//TODO: REMOVE MIGRATION ACTION AFTER INITIAL RELASE
tailf:action load-device-state {
  tailf:actionpoint pnp_load_device_state;
  input {
    list device {
      key serial;
      uses device-state;
    }
  }
  output {
  }
}

container server {
  leaf ip-address {
    type inet:ip-address {
      tailf:info "PNP Server ip address.";
    }
    default "0.0.0.0";
    description
      "The address on which the PnP server will listen
       for incoming requests. Default is to listen on
       all addresses available.";
  }

  leaf-list additional-ip-address {
    type inet:ip-address {
      tailf:info "Additional PnP Server IP address.";
    }
    description
      "Additional IP address on which the PnP server will listen
       for incoming requests.";
  }

  leaf port {
    type inet:port-number {
      tailf:info "PNP Server port.";
    }
    default 9191;
    description
      "The port on which the PnP server will listen
       for incoming requests.";
```

```
    }

    leaf use-ssl {
      type boolean {
        tailf:info "Enable or disable HTTPS.";
      }
      default true;
      description
        "If set to true, the PnP server will use the https
         protocol. If false, http will be used.";
    }

    leaf verify-cpe-cert {
      type boolean;
      default false;
      description
        "If set to true, the PnP server will do require client
         authentication. If false, no client authentication will
         be required.";
    }

    leaf ca-cert-file {
      type string;
      default "ca_cert";
      description
        "The CA certificate to use when doing client authentication";
    }

    leaf docroot {
      type string;
      description
        "Document root directory of the PnP server";
    }

    tailf:info "PNP Server configuration.";
  }

  container proxy-servers {
    tailf:info "Proxy Server Support.";

    description
      "If proxy servers are used in front of the PNP server the
       reported IP address of the device (as stored in
       /pnp-state/device/ip-address) will be the proxy IP address
       instead of the device IP address.

       If the proxy server supports forwarding the client IP address
       using the X-Forwarded-For header, the PNP server can be
       configured to use that as the client IP address.

       By default the PNP server will not honor the X-Forwarded-For
       header, it must be enabled here, either by listing the IP
       addresses of the proxies (in which case X-Forwarded-For will
       be honored only from those IP addresses) or by setting
       'allow-any' to 'true'.";

    choice x-forwarded-for {
      leaf allow-any {
        type empty;
        description
          "If set to true, then client IP address will be taken from
           the X-Forwarded-For header (if present).";
```

```
      tailf:info "Always take client IP from X-Forwarded-For header";
    }
    leaf-list allow-from {
      type inet:ip-address;
      description "Only honor X-Forwarded-For from these addresses.";
      tailf:info "Proxy Server addresses";
    }
  }
}

container secondary-server {
  presence "Enables the secondary PnP server";

  leaf ip-address {
    type inet:ip-address {
      tailf:info "PnP secondary server ip address.";
    }
    default "0.0.0.0";
    description
      "The address on which the secondary PnP server will listen
       for incoming requests. Default is to listen on
       all addresses available.";
  }

  leaf-list additional-ip-address {
    type inet:ip-address {
      tailf:info "Additional secondary PnP server IP address.";
    }
    description
      "Additional IP address on which the secondary PnP server will listen
       for incoming requests.";
  }

  leaf port {
    type inet:port-number {
      tailf:info "PnP secondary server port.";
    }
    default 9090;
    description
      "The port on which the secondary PnP server will listen
       for incoming requests.";
  }

  leaf docroot {
    type string;
    description
      "Document root directory of the secondary PnP server";
  }

  tailf:info "Secondary PnP Server configuration used for certificate
              installation.";
}

must "not(/pnp/server/port=/pnp/secondary-server/port)" {
    error-message "Primary and secondary server cannot have same port";
}

container logging {
  list serial {
    key name;
    leaf name {
      type string {
```

```
        tailf:info "UDI or all;;Name of UDI to log, 'all' will apply
                    to all CPEs.";
      }
    }
  }
  tailf:info "PNP Server logging configuration.";
}

list reverse-map {
  config false;
  tailf:cdb-oper {
    tailf:persistent true;
  }
  key serial;
  description
    "Reverse map from device serial number to pnp map id.";

  leaf serial {
     type string {
      tailf:info "WORD;;Serial number of device";
      length "1..32";
    }
  }
  leaf id {
    type string {
      tailf:info "WORD;;pnp map id for look-up";
    }
  }
}

list map {
  key id;
  unique serial;

  description
    "Map from device serial number to device name. This
     information is used when mounting the device into the
     NCS device tree.";

  leaf id {
    type string {
      tailf:info "WORD;;pnp map id";
    }
  }

  leaf serial {
    type string {
      tailf:info "WORD;;Serial number of device";
      length "1..32";
    }
    description
      "Serial number of device. The device sends this as
       part of the URI.";
  }

  leaf apply-certificate-install {
    type boolean;
    default false;
    description
      "Do certificate installation on the client when connecting
       on the secondary server.";
  }
```

```
leaf username {
  type string {
    tailf:info "WORD;;Username for configuring device";
  }
  mandatory true;
  description
    "The username that will be used when communicating using PnP
     with the device and for managing a device if no authgrup has
     been specified.";
}

leaf password {
  tailf:info
    "The password that will be used when communicating using PnP
     with the device and managing the device.";
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
}

leaf sec-password {
  tailf:info "Define the second password for the CPE.";
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
}

leaf nso-key {
  type tailf:aes-cfb-128-encrypted-string;
  tailf:suppress-echo true;
  description
    "Key that needs to be specified when requesting the device state
     using the device-state action. If not set allow the action
     regardless of the key supplied when executing the action.";
}

choice day0 {
  leaf day0-file {
    type string;
    default "CPE.txt";
    tailf:info "day0 config filename.";
    description
      "Filename of file containing the day0 config.";
  }
  leaf-list day0-template {
    type string;
    tailf:info "Day0 config template filename.";
    description
      "Template files to use for creating Day0 config for the device.";
    ordered-by user;
    min-elements 1;
  }
  default day0-file;
}

list day0-inline {
  tailf:info "day0 config inline.";
  key name;
  ordered-by user;

  leaf name {
    type string;
    tailf:info "Day0 config config/template filename.";
```

```
  }

  leaf inline {
    type string;
    tailf:info "day0 config inline.";
    mandatory true;
  }

  must "name = ../day0-file or name = ../day0-template" {
    error-message "Name must match day0-file or day0-template";
  }
}

container cfg-properties {
  uses variables;
}

leaf cfg-common {
  type leafref {
    path "../../day0-common/cfgName";
  }
}

leaf apply-config-upgrade {
  type boolean;
  default false;
  description
    "Set to true if pnp server need to send configuration to
     device through config-upgrade ";
}

tailf:info "Mapping between serial number and device.";

must "apply-certificate-install = 'false' or
      (apply-certificate-install = 'true' and
      /pnp/server/use-ssl = 'true')" {
  error-message "PnP server need use-ssl to be true when
                 apply-certificate-install is true.";
}

leaf sudi-authentication {
  type boolean;
  default false;
  description
    "Set to true if SUDI authentication is needed for the device";
}

leaf use-relative-url {
  type boolean;
  default false;
  description
    "Set to true if a relative URL should be used in config upgrade
     and certificate install work requests";
}

leaf need-clean {
  type empty;
  description
    "If set send Day--1 config to the device when it is deleted from the
     PnP map";
}
```

```
  uses config-restore {
    refine config-restore/config-restore-file {
      description
        "Name of file on CPE that will be used when restoring device to
         initial config.";
    }
    refine config-restore/config-restore-uri {
      description
        "URI of file that will be used when restoring device to
         initial config.";
    }
    refine config-restore/config-restore-disabled {
      description
        "Disable config restore.";
    }
  }
}

list day0-common {
  key cfgName;
  leaf cfgName {
    type string;
    description
      "Name of the common variables to be shared for pnp devices";
  }
  uses variables;
}

leaf cfg-location {
  type string;
  description
    "optional, cfg files location. if it absent, it will be cfg
     folder of cisco-pnp package";
}

leaf wait-after-reload-time {
  tailf:info "Wait time between issuing reload and applying day0
              configuration for CPE";
  type uint32 {
    tailf:info "<INT>;;Wait time in seconds between reloading and
                applying day0 for CPE configuration.";
  }
  default "20";
}

container state-value {
  leaf secure-pnp {
    description
      "Enable secure PnP which will make the PnP server never send out
       usr/pwd to a device before the device has completed SUDI
       authentication. For this setting to work one has to enable SUDI
       for all devices otherwise most PnP work requests will fail since
       the PnP server won't send out the username and password required
       to pass AAA on devices.";
    type boolean;
    default true;
  }

  leaf backoff-timeout {
    tailf:info "Timeout between CPE re-sync with the PnP. This parameter
                controls how long it will take for the PnP server to
                notice that a device has changed ip address.";
```

```
      type uint32 {
        tailf:info "<INT>;;Timeout in seconds between client reconnect.";
      }
      default "210";
    }

    leaf terminate-when-done {
      type empty;
      description
        "If present PnP process will be terminated and CPE will no longer
         re-sync with PnP server after the device onboard.";
    }
  }

  list unclaimed {
    config false;
    key id;
    leaf id {
      type string;
    }

    tailf:info "List of devices without a mapping from serial to device.";
    description
      "List of unclaimed devices, ie devices that does not have
       a mapping in the map table above. If a map entry is added
       then the device will be moved from this list the next time
       it contacts the pnp server.";
  }

  list device-map {
    key expr;

    leaf expr {
      type string;
      tailf:info "Reg expr used to match against device-info versionstring.";
      description
        "Regular expression to match agains device-info versionstring.";
    }

    leaf wan {
      type string;
      mandatory true;
      tailf:info "WAN interface on CPE device.";
      description
        "WAN interface on CPE device.";
    }

    leaf lan {
      type string;
      mandatory true;
      tailf:info "LAN interface on CPE device.";
      description
        "LAN interface on CPE device.";
    }

    uses config-restore {
      refine config-restore {
        default config-restore-file;
      }
      refine config-restore/config-restore-file {
        description
          "Name of file on cpe that will be used when restoring device to
```

```
            initial config. The pnp server will send the commands: \n
            copy <config-restore-file> startup-config \n reload";
        }
        refine config-restore/config-restore-uri {
          description
            "URI of file that will be used when restoring device to
             initial config. The pnp server will send the commands: \n
             copy <config-restore-uri> startup-config \n reload";
        }
        refine config-restore/config-restore-disabled {
          description
            "Disable config restore.";
        }
      }

      leaf trustpoint-label {
        tailf:info "Label to use for the trustpoint in certificate install.";
        description
          "Label to use for the trustpoint in certificate install.";
        type union {
          type string;
          type enumeration {
            enum random;
          }
        }
      }

      choice server-side-certificate-choice {
        leaf server-side-certificate {
          type string;
          tailf:info
            "Certificate (from file) to be sent to the PnP agent in the certificate
             install request.";
          description
            "Certificate to be sent to the PnP agent in the certificate
             install request. This is not mandatory, if this leaf is not
             set, the Server host certificate will be sent.";
        }
        leaf server-side-certificate-inline {
          type string;
          tailf:info
            "Certificate in-line to be sent to the PnP agent in the certificate
             install request.";
          description
            "Certificate to be sent to the PnP agent in the certificate
             install request. This is not mandatory, if this leaf is not
             set, the Server host certificate will be sent.";
        }
      }

      leaf disable-serial-check {
        type empty;
        tailf:info "If present will disable the SUDI Serial check.";
        description "If present will disable the SUDI Serial check.  This will
        prevent matching the SUDI with the serial, however the SUDI key will still
        be verified.";
      }
    }
  }

  container pnp-state {
    config false;
```

```
    tailf:cdb-oper {
      tailf:persistent true;
    }

    leaf version {
      type string;
      default "3.0.0";
    }

    list device {
      key serial;
      uses device-state;
    }
  }

}
```

NSO Cisco PnP Server Notif Model

Example 7. NSO Cisco PnP Server Notification YANG Model

```
module cisco-confd-pnp-notif {
  namespace "http://cisco.com/ns/nso/cfp/pnp-server-notif";
  prefix "confd-pnp-notif";

  import ietf-inet-types { prefix "inet"; }

  organization "Cisco Systems";
  description "CONFD PNP Server";

  revision 2019-10-25 {
    description "Initial revision";
  }

  grouping device-notification {
    leaf serial {
      type string;
    }
    leaf id {
      type string;
    }
    leaf udi {
      type string;
    }
  }

  notification device-added {
    uses device-notification;
    leaf ip-address {
      type inet:ip-address;
    }
    leaf port {
      type inet:port-number;
    }
    leaf device-info {
      type string;
    }
    leaf host-name {
      type string;
    }
  }
```

```
notification device-updated {
  uses device-notification;
  leaf ip-address {
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
  leaf host-name {
    type string;
  }
}

notification device-deleted {
  uses device-notification;
}

notification device-connected {
  uses device-notification;
  leaf ip-address {
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
  leaf device-info {
    type string;
  }
  leaf host-name {
    type string;
  }
  leaf is-netsim {
    type boolean;
  }
  leaf discovery-created {
    type boolean;
  }
}

notification unclaimed-list {
  leaf serial {
    type string;
  }
  leaf udi {
    type string;
  }
  leaf is-netsim {
    type boolean;
  }
  leaf operation {
    type string;
  }
}

notification error {
  uses device-notification;

  leaf request {
    type string;
  }
  leaf error-message {
```

```
        type string;
      }
    }
  }
```

CHAPTER 7

Resources

References for further reading

NSO Packages chapter in NSO 5.2.3 Administration Guide.

The AAA Infrastructure chapter in NSO 5.2.3 Administration Guide.

Embedded Erlang applications chapter in the NSO 5.2.3 Development guide.

PnP Map Details

The details below document the specifics of the PnP map, this is identical to the information shown in the NSO CLI.

pnp/map

apply-certificate-install	Do certificate installation on the client when connecting on the secondary server.
apply-config-upgrade	Set to true if PNP server needs to send configuration to device through config-upgrade.
authgroup	Credentials to use when managing the device.
auto-update-mgmt-ip	If source PNP client source IP changes, the source IP will be used to update the management IP address.
commit-queue-temp	Use commit-queue for the device if true.
config-erase	Erase the configuration on the device.
config-restore-disabled	Disable the config restore feature.
config-restore-file	Restore configuration from the given file.
config-restore-uri	Restore configuration from the given URI.
day0/cfg-common	Use a common day0 configuration.
day0/cfg-properties	Device specific template variables.
day0/day0-file	Day0 config filename.
device-name	Name to mount the device under in NCS.
device-type	The device-type defines how the communication towards the device is carried out.

managed	The device will be added to CDB device tree.
mgmt-ip-address	The management ip address of the device.
ned-id	NED id to use when mounting device.
need-clean	If set, send Day--1 config to the device when it is deleted from the PnP map.
nso-key	Key that needs to be specified when requesting the device state using the device-state action.
password	The password that will be used when communicating using PnP with the device and managing the device.
port	Management port for the device.
sec-password	Define the secondary password for the CPE.
serial	Serail number of the device.
sudi-authentication	Set to true if SUDI authentication is needed for the device.
use-hostname	Name from device PNP protocol to mount the device under in NSO.
use-relative-url	Relative URL should be used in config upgrade and certificate install work requests.
username	Username for configuring device.