



NSO 4.4.2.3 Manual Pages

First Published: May 17, 2010

Last Modified: September 1, 2017

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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <http://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

This product includes software developed by the NetBSD Foundation, Inc. and its contributors.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit <http://www.openssl.org/>.

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

U.S. Pat. No. 8,533,303 and 8,913,519

Copyright © 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017 Cisco Systems, Inc. All rights reserved.



CONTENTS

CHAPTER I

NCS man-pages, Volume 1 1

 nsc-backup 3

 nsc-collect-tech-report 5

 nsc-installer 7

 nsc-maapi 11

 nsc-make-package 17

 nsc-netsim 21

 nsc-project-create 25

 nsc-project-export 27

 nsc-project-git 29

 nsc-project-setup 31

 nsc-project-update 33

 nsc-project 35

 nsc-setup 37

 nsc-uninstall 41

 nsc 43

 nsc_cli 47

 nsc_cmd 51

 nsc_load 53

 nscsc 59

 nct-backup 71

 nct-check 75

 nct-cli-cmd 79

 nct-copy 83

 nct-get-logs 87

 nct-ha 91

 nct-hostsfile 95

 nct-install 97

nct-load-config	101
nct-move-device	105
nct-packages	109
nct-patch	113
nct-ssh-cmd	117
nct-start	121
nct-stop	125
nct-upgrade	129
nct	133
pyang	143

CHAPTER II

NCS man-pages, Volume 3	155
confd_lib	157
confd_lib_cdb	159
confd_lib_dp	197
confd_lib_events	255
confd_lib_ha	261
confd_lib_lib	263
confd_lib_maapi	285
confd_types	351

CHAPTER III

NCS man-pages, Volume 5	387
clispec	389
mib_annotations	431
ncs.conf	433
tailf_yang_cli_extensions	465
tailf_yang_extensions	505

NCS man-pages, Volume 1



Name

ncs-backup — Command to backup and restore NCS data

Synopsis

```
ncs-backup [--install-dir InstallDir] [--no-compress]
```

```
ncs-backup --restore [Backup] [--install-dir InstallDir] [--non-interactive]
```

DESCRIPTION

The **ncs-backup** command can be used to backup and restore NCS CDB, state data, and config files for an NCS "system installation", i.e. one that was done with the `--system-install` option to the NCS installer (see [ncs-installer\(1\)](#)). Unless the `--restore` option is used, the command creates a backup. The backup is stored in the *RunDir/backups* directory, named with the NCS version and current date and time.

OPTIONS

<code>--restore [<i>Backup</i>]</code>	Restore a previously created backup. The <i>Backup</i> argument is either the name of a file in the <i>RunDir/backups</i> directory or the full path to a backup file. If the argument is omitted, unless the <code>--non-interactive</code> option is given, the command will offer selection from available backups.
<code>[--install-dir <i>InstallDir</i>]</code>	Specifies the directory for installation of NCS static files, like the <code>--install-dir</code> option to the installer. If this option is omitted, <code>/opt/ncs</code> will be used for <i>InstallDir</i> .
<code>[--non-interactive]</code>	If this option is used, restore will proceed without asking for confirmation.
<code>[--no-compress]</code>	If this option is used, the backup will not be compressed (default is compressed). The restore will uncompress if the backup is compressed, regardless of this option.



Name

`ncs-collect-tech-report` — Command to collect diagnostics from an NCS installation that has been installed with the `--system-install` option

Synopsis

```
ncs-collect-tech-report [--install-dir InstallDir] [--full] [--num-debug-dumps Integer]
```

DESCRIPTION

The **ncs-collect-tech-report** command can be used to collect diagnostics from an NCS installation. The resulting diagnostics file contains information that is useful to Tail-f support to diagnose problems and errors.

If the NCS daemon is running, runtime data from the running daemon will be collected. If the NCS daemon is not running, only static files will be collected.

OPTIONS

[<code>--install-dir</code> <i>InstallDir</i>]	Specifies the directory for installation of NCS static files, like the <code>--install-dir</code> option to the installer. If this option is omitted, <code>/opt/ncs</code> will be used for <i>InstallDir</i> .
[<code>--full</code>]	This option is used to also include a full backup (as produced by ncs-backup) of the system. This makes it sometimes possible for Tail-f support to reproduce issues locally.
[<code>--num-debug-dumps</code> <i>Count</i>]	This option is sometimes useful when a resource leak (memory/file descriptors) is suspected. It instructs the ncs-collect-tech-report script to run the command ncs --debug-dump multiple times.



Name

ncs-installer — NCS installation script

Synopsis

```
ncs-VSN.OS.ARCH.installer.bin [--local-install] LocalInstallDir
```

```
ncs-VSN.OS.ARCH.installer.bin --system-install [--install-dir InstallDir] [--config-dir ConfigDir] [--run-dir RunDir] [--log-dir LogDir] [--run-as-user User] [--keep-ncs-setup] [--non-interactive]
```

DESCRIPTION

The NCS installation script can be invoked to do either a simple "local installation", which is convenient for test and development purposes, or a "system installation", suitable for deployment.

LOCAL INSTALLATION

```
[ --local-install ]  
LocalInstallDir
```

When the NCS installation script is invoked with this option, or is given only the *LocalInstallDir* argument, NCS will be installed in the *LocalInstallDir* directory only.

SYSTEM INSTALLATION

```
--system-install
```

When the NCS installation script is invoked with this option, it will do a system installation that uses several different directories, in accordance with Unix/Linux application installation standards. The first time a system installation is done, the following actions are taken:

- The directories described below are created and populated.
- An init script for start of NCS at system boot is installed.
- User profile scripts that set up \$PATH and other environment variables appropriately for NCS users are installed.
- A symbolic link that makes the installed version the currently active one is created (see the --install-dir option).

```
[ --install-dir  
InstallDir ]
```

This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is *InstallDir*/ncs-VSN, allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link *InstallDir*/current pointing to one of the ncs-VSN directories. If the --install-dir option is omitted, /opt/ncs will be used for *InstallDir*.

```
[ --config-dir ConfigDir  
]
```

This directory is used for config files, e.g. ncs.conf. If the --config-dir option is omitted, /etc/ncs will be used for *ConfigDir*.

```
[ --run-dir RunDir ]
```

This directory is used for run-time state files, such as the CDB data base and currently used packages. If the --run-dir option is omitted, /var/opt/ncs will be used for *RunDir*.

[--log-dir *LogDir*]

This directory is used for the different log files written by NCS. If the --log-dir option is omitted, /var/log/ncs will be used for *LogDir*.

[--run-as-user *User*]

By default, the system installation will run NCS as the root user. If a different user is given via this option, NCS will instead be run as that user. The user will be created if it does not already exist. This mode is only supported on Linux systems that have the **setcap** command, since it is needed to give NCS components the required capabilities for some aspects of the NCS functionality.

When the option is used, the following executable files (assuming that the default /opt/ncs is used for --install-dir) will be installed with elevated privileges:

/opt/ncs/current/lib/
ncs/lib/core/pam/priv/
epam

Setuid to root. This is typically needed for PAM authentication to work with a local password file. If PAM authentication is not used, or if the local PAM configuration does not require root privileges, the setuid-root privilege can be removed by using **chmod u-s**.

/opt/ncs/current/lib/
ncs/erts/bin/ncs /opt/
ncs/current/lib/ncs/
erts/bin/ncs.smp

Capability
cap_net_bind_service. One of these files (normally ncs.smp) will be used as the NCS daemon. The files have execute access restricted to the user given via --run-as-user. The capability is needed to allow the daemon to bind to ports below 1024 for northbound access, e.g. port 443 for HTTPS or port 830 for NETCONF over SSH. If this functionality is not needed, the capability can be removed by using **setcap -r**.

/opt/ncs/current/lib/
ncs/bin/ip

Capability
cap_net_admin. This is a copy of the OS **ip(8)** command, with execute access restricted to the user given via --run-as-user. The program is not used by the core NCS daemon, but provided for packages that need to configure IP addresses on interfaces (such as the **tailf-hcc** package).

/opt/ncs/current/lib/
ncs/bin/arping

If no such packages are used, the file can be removed.
Capability `cap_net_raw`. This is a copy of the OS **arping(8)** command, with execute access restricted to the user given via `--run-as-user`. The program is not used by the core NCS daemon, but provided for packages that need to send gratuitous ARP requests (such as the `tailf-hcc` package). If no such packages are used, the file can be removed.



Note

When the `--run-as-user` option is used, all OS commands executed by NCS will also run as the given user, rather than as the user specified in e.g. [clispec\(5\)](#) definitions for custom CLI commands.

[`--keep-ncs-setup`]

The **ncs-setup** command is not usable in a "system installation", and is therefore by default excluded from such an installation to avoid confusion. This option instructs the installation script to include **ncs-setup** in the installation despite this.

[`--non-interactive`]

If this option is given, the installation script will proceed with potentially disruptive changes (e.g. modifying or removing existing files) without asking for confirmation.



Name

ncs-maapi — command to access an ongoing transaction

Synopsis

```
ncs-maapi --get Path...
ncs-maapi --set Path Value [ Path Value... ]
ncs-maapi --keys Path...
ncs-maapi --exists Path...
ncs-maapi --delete Path...
ncs-maapi --create Path...
ncs-maapi --insert Path...
ncs-maapi --revert
ncs-maapi --msg To Message Sender
--priomsg To Message
--sysmsg To Message
ncs-maapi --cliget Param...
ncs-maapi --cliset Param Value [ Param Value... ]
ncs-maapi --cmd2path Cmd [ Cmd ]
ncs-maapi --cmd-path [--is-deleta ] [--emit-parents ] [--non-recursive ] Path [ Path ]
ncs-maapi --cmd-diff Path [ Path ]
ncs-maapi --keypath-diff Path
ncs-maapi --clcmd [--get-io ] [--no-hidden ] [--no-error ] [--no-aaa ] [--no-fullpath ] [--unhide
<group>] Cli command...
```

DESCRIPTION

This command is intended to be used from inside a CLI command or a NETCONF extension RPC. These can be implemented in several ways, as an action callback or as an executable.

It is sometimes convenient to use a shell script to implement a CLI command and then invoke the script as an executable from the CLI. The ncs-maapi program makes it possible to manipulate the transaction in which the script was invoked.

Using the ncs-maapi command it is possible to, for example, write configuration wizards and custom show commands.

OPTIONS

-g, --get <i>Path</i> ...	Read element value at <i>Path</i> and display result. Multiple values can be read by giving more than one <i>Path</i> as argument to get.
-s, --set <i>Path Value</i> ...	Set the value of <i>Path</i> to <i>Value</i> . Multiple values can be set by giving multiple <i>Path Value</i> pairs as arguments to set.
-k, --keys <i>Path</i> ...	Display all instances found at <i>path</i> . Multiple <i>Paths</i> can be specified.

<code>-e, --exists Path ...</code>	Exit with exit code 0 if Path exists (if multiple paths are given all must exist for the exit code to be 0).
<code>-d, --delete Path ...</code>	Delete element found at Path.
<code>-c, --create Path ...</code>	Create the element Path.
<code>-i, --insert Path ...</code>	Insert the element at Path. This is only possible if the elem has the 'indexed-view' attribute set.
<code>-z, --revert</code>	Remove all changes in the transaction.
<code>-m, --msg To Message Sender</code>	Send message to a user logged on to the system.
<code>-Q, --priomsg To Message</code>	Send prio message to a user logged on to the system.
<code>-M, --sysmsg To Message</code>	Send system message to a user logged on to the system.
<code>-G, --cliget Param ...</code>	Read and display CLI session parameter or attribute. Multiple params can be read by giving more than one Param as argument to cliget. Possible params are complete-on-space, idle-timeout, ignore-leading-space, paginate, "output file", "screen length", "screen width", terminal, history, autowizard, "show defaults", and if enabled, display-level. In addition to this the attributes called annotation, tags and inactive can be read.
<code>-S, --cliset Param Value ...</code>	Set CLI session parameter to Value. Multiple params can be set by giving more than one Param-Value pair as argument to cliset. Possible params are complete-on-space, idle-timeout, ignore-leading-space, paginate, "output file", "screen length", "screen width", terminal, history, autowizard, "show defaults", and if enabled, display-level.
<code>-E, --cmd-path [--is- delete] [--emit-parents] [--non-recursive] Path</code>	Display the C- and I-style command for a given path. Optionally display the command to delete the path, and optionally emit the parents, ie the commands to reach the submode of the path.
<code>-L, --cmd-diff Path</code>	Display the C- and I-style command for going from the running configuration to the current configuration.
<code>-q, --keypath-diff Path</code>	Display the difference between the current state in the attached transaction and the running configuration. One line is emitted for each difference. Each such line begins with the type of the change, followed by a colon (':') character and lastly the keypath. The type of the change is one of the following: "created", "deleted", "modified", "value set", "moved after" and "attr set".
<code>-T, --cmd2path Cmd</code>	Attempts to derive an aaa-style namespace and path from a C-/I-style command path.
<code>-C, --clcmd [--get- io] [--no-hidden] [--no- error] [--no-aaa] [--no- fullpath] [--unhide group] Cli command to execute</code>	Execute cli command in ongoing session, optionally ignoring that a command is hidden, unhiding a specific hide group, or ignoring the fullpath check of the argument to the show command. Multiple hide groups may be unhidden using the --unhide parameter multiple times.

EXAMPLE

Suppose we want to create an add-user wizard as a shell script. We would add the command in the clispec file `ncs.cli` as follows:

```
...
<configureMode>
  <cmd name="wizard">
```



```

    <info>Configuration wizards</info>
    <help>Configuration wizards</help>
    <cmd name="adduser">
        <info>Create a user</info>
        <help>Create a user</help>
        <callback>
            <exec>
                <osCommand>./adduser.sh</osCommand>
            </exec>
        </callback>
    </cmd>
</cmd>
</configureMode>
...

```

And have the following script `adduser.sh`:

```

#!/bin/bash

## Ask for user name
while true; do
    echo -n "Enter user name: "
    read user

    if [ ! -n "${user}" ]; then
        echo "You failed to supply a user name."
    elif ncs-maapi --exists "/aaa:aaa/authentication/users/user${user}"; then
        echo "The user already exists."
    else
        break
    fi
done

## Ask for password
while true; do
    echo -n "Enter password: "
    read -s pass1
    echo

    if [ "${pass1:0:1}" == "$" ]; then
        echo -n "The password must not start with $. Please choose a "
        echo "different password."
    else
        echo -n "Confirm password: "
        read -s pass2
        echo

        if [ "${pass1}" != "${pass2}" ]; then
            echo "Passwords do not match."
        else
            break
        fi
    fi
done

groups=`ncs-maapi --keys "/aaa:aaa/authentication/groups/group"`
while true; do
    echo "Choose a group for the user."
    echo -n "Available groups are: "
    for i in ${groups}; do echo -n "${i} "; done
    echo
    echo -n "Enter group for user: "
    read group

```

```

        if [ ! -n "${group}" ]; then
echo "You must enter a valid group."
        else
for i in ${groups}; do
        if [ "${i}" == "${group}" ]; then
# valid group found
break 2;
        fi
done
echo "You entered an invalid group."
        fi
echo
done

echo
echo "Creating user"
echo
ncs-maapi --create "/aaa:aaa/authentication/users/user${user}"
ncs-maapi --set "/aaa:aaa/authentication/users/user${user}/password" \
"${pass1}"

echo "Setting home directory to: /var/ncs/homes/${user}"
ncs-maapi --set "/aaa:aaa/authentication/users/user${user}/homedir" \
"/var/ncs/homes/${user}"

echo

echo "Setting ssh key directory to: "
echo "/var/ncs/homes/${user}/ssh_keydir"
ncs-maapi --set "/aaa:aaa/authentication/users/user${user}/ssh_keydir" \
"/var/ncs/homes/${user}/ssh_keydir"

echo

ncs-maapi --set "/aaa:aaa/authentication/users/user${user}/uid" "1000"
ncs-maapi --set "/aaa:aaa/authentication/users/user${user}/gid" "100"

echo "Adding user to the ${group} group."
gusers=`ncs-maapi --get "/aaa:aaa/authentication/groups/group${group}/users" `

for i in ${gusers}; do
        if [ "${i}" == "${user}" ]; then
echo "User already in group"
exit 0
        fi
done
ncs-maapi --set "/aaa:aaa/authentication/groups/group${group}/users" \
"${gusers} ${user}"

echo
exit 0

```

DIAGNOSTICS

On success exit status is 0. On failure 1 or 2. Any error message is printed to stderr.

ENVIRONMENT VARIABLES

Environment variables are used for determining which user session and transaction should be used when performing the operations. The NCS_MAAPI_USID and NCS_MAAPI_THANDLE environment variables are automatically set by NCS when invoking a CLI command, but when a NETCONF extension RPC is invoked, only NCS_MAAPI_USID is set, since there is no transaction associated with such an invocation.

NCS_MAAPI_USID	User session to use.
NCS_MAAPI_THANDLE	The transaction to use when performing the operations.
NCS_MAAPI_DEBUG	Maapi debug information will be printed if this variable is defined.
NCS_IPC_ADDR	The address used to connect to the NSO daemon, overrides the compiled in default.
NCS_IPC_PORT	The port number to connect to the NSO daemon on, overrides the compiled in default.

SEE ALSO

The NSO User Guide

ncs(1) - command to start and control the NSO daemon

ncsc(1) - YANG compiler

ncs.conf(5) - NSO daemon configuration file format

clispec(5) - CLI specification file format



Name

ncs-make-package — Command to create an NCS package

Synopsis

ncs-make-package [OPTIONS] package-name

DESCRIPTION

Creates an NCS package of a certain type. For NEDs, it creates a netsim directory by default, which means that the package can be used to run simulated devices using ncs-netsim, i.e that ncs-netsim can be used to run simulation network that simulates devices of this type.

The generated package should be seen as an initial package structure. Once generated, it should be manually modified when it needs to be updated. Specifically, the package-meta-data.xml file must be modified with correct meta data.

OPTIONS

-h, --help	Print a short help text and exit.
--dest Directory	By default the generated package will be written to a directory in current directory with the same name as the provided package name. This optional flag writes the package to the --dest provided location.
--build	Once the package is created, build it too.
--snmp-ned Directory	Create a SNMP NED package, using the device MIB files in Directory.
--netconf-ned Directory	Create a NETCONF NED package, using the device YANG files in DIR.
--service-skeleton java java-and-template python python-and-template template	Generate a skeleton package for a simple RFS service, either implemented by Java code, Python code, based on a template, or a combination of them.
--generic-ned-skeleton	Generate a skeleton package for a generic NED. This is a good starting point whenever we wish to develop a new generic NED.
--data-provider-skeleton	Generate a skeleton package for a simple data provider.
--erlang-skeleton	Generate a skeleton for an Erlang package.

SERVICE SPECIFIC OPTIONS

--augment PATH	Augment the generated service model under PATH, e.g. /ncs:services.
--root-container NAME	Put the generated service model in a container named NAME.

JAVA SPECIFIC OPTIONS

--java-package NAME	NAME is the Java package name for the Java classes generated from all device YANG modules. These classes can be used by Java code implementing for example services.
------------------------	--

NED SPECIFIC OPTIONS

<code>--no-netsim</code>	Do not generate a netsim directory. This means the package cannot be used by ncs-netsim.
<code>--no-java</code>	Do not generate any Java classes from the device YANG modules.
<code>--vendor</code> VENDOR	The vendor element in the package file.

NETCONF NED SPECIFIC OPTIONS

<code>--pyang-sanitize</code>	Sanitize the device's YANG files. This will invoke pyang --sanitize on the device YANG files.
<code>--confd-netsim-db-mode</code> candidate startup running-only	Control which datastore netsim should use when simulating the device. The candidate option here is default and it includes the setting writable-through-candidate
<code>--ncs-depend-package</code> DIR	If the yang code in a package depends on the yang code in another NCS package we need to use this flag. An example would be if a device model augments YANG code which is contained in another NCS package. The arg, the package we depend on, shall be relative the src directory to where the package is built.

PYTHON SPECIFIC OPTIONS

<code>--component-class</code> module.Class	This optional parameter specifies the <i>python-class-name</i> of the generated package-meta-data.xml file. It must be in format <i>module.Class</i> . Default value is <i>main.Main</i> .
<code>--action-example</code>	This optional parameter will produce an example of an Action.
<code>--subscriber-example</code>	This optional parameter will produce an example of a CDB subscriber.

ERLANG SPECIFIC OPTIONS

<code>--erlang-application-</code> name NAME	Add a skeleton for an Erlang application. Invoke the script multiple times to add multiple applications.
---	--

LSA SPECIFIC OPTIONS

<code>--lsa</code> PATH BASE-NAME	Create an LSA dispatcher based on the YANG file at PATH. The name of the package will be <BASE-NAME>.
<code>--lsa-upper</code> PATH BASE-NAME	Create an LSA NED to be loaded on the upper node. The name of the package will be lsa-upper-<BASE-NAME>. The name of the module will be lsa-<BASE-NAME> and the prefix lsa-<BASE-NAME>. The rewritten YANG file can be found in src/lsa-out of the package when the package has successfully been built.
<code>--lsa-lower</code> PATH TYPE BASE-NAME	Create an LSA service package of TYPE to be loaded on the lower node. The name of the package will be lsa-lower-<BASE-NAME> The name of the YANG module will be lsa-<BASE-NAME> and the prefix lsa-<BASE-NAME>. The rewritten YANG file

can be found in `src/lsa-out` of the package when the package has successfully been built.

EXAMPLES

Generate a NETCONF NED package given a set of YANG files from a fictitious acme router device.

```
$ ncs-make-package --netconf-ned /path/to/yangfiles acme  
$ cd acme/src; make all
```

This package can now be used by `ncs-netsim` to create simulation networks with simulated acme routers.



Name

ncs-netsim — Command to create and manipulate a simulated network

Synopsis

```
ncs-netsim create-network NcsPackage NumDevices Prefix [ --dir NetsimDir ]  
ncs-netsim add-to-network NcsPackage NumDevices Prefix [ --dir NetsimDir ]  
ncs-netsim delete-network [ --dir NetsimDir ]  
ncs-netsim start | stop | is-alive | reset | restart | status [ Devicename ] [ --dir NetsimDir ] [ --  
  async / -a ]  
ncs-netsim netconf-console Devicename [ XPathFilter ] [ --dir NetsimDir ]  
ncs-netsim -w | --window cli | cli-c | cli-i Devicename [ --dir NetsimDir ]  
ncs-netsim list | ncs-xml-init | packages | whichdir [ --dir NetsimDir ]
```

DESCRIPTION

ncs-netsim is a script to create, control and manipulate simulated networks of managed devices. It is a tool targeted at NCS application developers. Each network element is simulated by ConfD, a Tail-f tool that acts as a NETCONF server, a Cisco CLI engine, or an SNMP agent.

OPTIONS

Commands

create-network
NcsPackage NumDevices
Prefix

Is used to create a new simulation network. The simulation network is written into a directory. This directory contains references to NCS packages that are used to emulate the network. These references are in the form of relative filenames, thus the simulation network can be moved as long as the packages that are used in the network are also moved.

This command can be given multiple times in one invocation of **ncs-netsim**. The mandatory parameters are:

- 1 *NcsPackage* is a either directory where an NCS NED package (that supports netsim) resides. Alternatively, just the name of one of the packages in `$NCS_DIR/packages/neds` can be used. Alternatively the *NcsPackage* is tar.gz package.
- 2 *NumDevices* indicates how many devices we wish to have of the type that is defined by the NED package.
- 3 *Prefix* is a string that will be used as prefix for the name of the devices

add-to-network
NcsPackage NumDevices
Prefix

Is used to add additional devices to a previously existing simulation network. This command can be given multiple times. The mandatory parameters are the same as for create-network.

**Note**

If we have already started NCS with an XML initialization file for the existing network, an updated initialization file will not take effect unless we remove the CDB database files, losing all NCS configuration. But we can replace the original initialization data with data for the complete new network when we have run `add-to-network`, by using **ncs_load** while NCS is running, e.g. like this:

`delete-network`

```
$ ncs-netsim ncs-xml-init > devices.xml
$ ncs_load -l -m devices.xml
```

Completely removes an existing simulation network. The devices are stopped, and the network directory is removed along with all files and directories inside it.

This command does not do any search for the network directory, but only uses `./netsim` unless the `--dir NetsimDir` option is given. If the directory does not exist, the command does nothing, and does not return an error. Thus we can use it in e.g. scripts or Makefiles to make sure we have a clean starting point for a subsequent `create-network` command.

`start [DeviceName]`

Is used to start the entire network, or optionally the individual device called `DeviceName`

`stop [DeviceName]`

Is used to stop the entire network, or optionally the individual device called `DeviceName`

`is-alive [DeviceName]`

Is used to query the 'liveness' of the entire network, or optionally the individual device called `DeviceName`

`status [DeviceName]`

Is used to check the status of the entire network, or optionally the individual device called `DeviceName`.

`reset [DeviceName]`

Is used to reset the entire network back into the state it was before it was started for the first time. This means that the devices are stopped, and all cdb files, log files and state files are removed. The command can also be performed on an individual device `DeviceName`.

`restart [DeviceName]`

This is the equivalent of 'stop', 'reset', 'start'

`-w | -window, cli | cli-c | cli-i DeviceName`

Invokes the ConfD CLI on the device called `DeviceName`. The flavor of the CLI will be either of Juniper (default) Cisco IOS (`cli-i`) or Cisco XR (`cli-c`). The `-w` option creates a new window for the CLI

`whichdir`

When we create the netsim environment with the `create-network` command, the data will by default be written into the `./netsim` directory unless the `--dir NetsimDir` is given.

All the control commands to stop, start, etc., the network need access to the netsim directory where the netsim data resides. Unless the `--dir NetsimDir` option is given we will search for the netsim directory in `$PWD`, and if not found there go upwards in the directory hierarchy until we find a netsim directory.

`list`

This command prints the result of that netsim directory search.

The netsim directory that got created by the `create-network` command contains a static file (by default `./`)

```
netconf-console  
DeviceName [XPathFilter]
```

```
ncs-xml-init  
[DeviceName]
```

```
packages
```

netconf-console) - this command prints the file content formatted. This command thus works without the network running.

Invokes the **netconf-console** NETCONF client program towards the device called DeviceName. This is an easy way to get the configuration from a simulated device in XML format.

Usually the purpose of running **ncs-netsim** is that we wish to experiment with running NCS towards that network. This command produces the XML data that can be used as initialization data for NCS and the network defined by this ncs-netsim installation.

List the NCS NED packages that were used to produce this ncs-netsim network.

Common options

```
--dir NetsimDir
```

When we create a network, by default it's created in `./netsim`. When we invoke the control commands, the netsim directory is searched for in the current directory and then upwards. The `--dir` option overrides this and creates/searches and instead uses NetsimDir for the netsim directory.

```
--async | -a
```

The start, stop, restart and reset commands can use this additional flag that runs everything in the background. This typically reduces the time to start or stop a netsim network.

EXAMPLES

To create a simulation network we need at least one NCS NED package that supports netsim. An NCS NED package supports netsim if it has a `netsim` directory at the top of the package. The NCS distribution contains a number of packages in `$NCS_DIR/packages/neds`. So given those NED packages, we can create a simulation network that use ConfD, together with the YANG modules for the device to emulate the device.

```
$ ncs-netsim create-network $NCS_DIR/packages/neds/c7200 3 c \  
create-network $NCS_DIR/packages/neds/nexus 3 n
```

The above command creates a test network with 6 routers in it. The data as well the execution environment for the individual ConfD devices reside in (by default) directory `./netsim`. At this point we can start/stop/control the network as well as the individual devices with the ncs-netsim control commands.

```
$ ncs-netsim -a start  
DEVICE c0 OK STARTED  
DEVICE c1 OK STARTED  
DEVICE c2 OK STARTED  
DEVICE n0 OK STARTED  
DEVICE n1 OK STARTED  
DEVICE n2 OK STARTED
```

Starts the entire network.

```
$ ncs-netsim stop c0
```

Stops the simulated router named `c0`.

```
$ ncs-netsim cli n1
```

Starts a Juniper CLI towards the device called `n1`.

ENVIRONMENT VARIABLES

- *NETSIM_DIR* if set, the value will be used instead of the `--dir Netsimdir` option to search for the netsim directory containing the environment for the emulated network
Thus, if we always use the same netsim directory in a development project, it may make sense to set this environment variable, making the netsim environment available regardless of where we are in the directory structure.
- *IPC_PORT* if set, the ConfD instances will use the indicated number and upwards for the local IPC port. Default is 5010. Use this if your host occupies some of the ports from 5010 and upwards.
- *NETCONF_SSH_PORT* if set, the ConfD instances will use the indicated number and upwards for the NETCONF ssh (if configured in `confd.conf`) Default is 12022. Use this if your host occupies some of the ports from 12022 and upwards.
- *NETCONF_TCP_PORT* if set, the ConfD instances will use the indicated number and upwards for the NETCONF tcp (if configured in `confd.conf`) Default is 13022. Use this if your host occupies some of the ports from 13022 and upwards.
- *SNMP_PORT* if set, the ConfD instances will use the indicated number and upwards for the SNMP udp traffic. (if configured in `confd.conf`) Default is 11022. Use this if your host occupies some of the ports from 11022 and upwards.
- *CLI_SSH_PORT* if set, the ConfD instances will use the indicated number and upwards for the CLI ssh traffic. (if configured in `confd.conf`) Default is 10022. Use this if your host occupies some of the ports from 10022 and upwards.

The `ncs-setup` tool will use these numbers as well when it generates the init XML for the network in the `ncs-netsim` network.

Name

`ncs-project-create` — Command to create an NCS project

Synopsis

`ncs-project create [OPTIONS] project-name`

DESCRIPTION

Creates an NCS project, which consists of directories, configuration files and packages necessary to run an NCS system.

After running this command, the command: `ncs-project update`, should be run.

The NCS project connects an NCS installation with an arbitrary number of packages. This is declared in a `project-meta-data.xml` file which is located in the directory structure as created by this command.

The generated project should be seen as an initial project structure. Once generated, the `project-meta-data.xml` file should be manually modified. After the `project-meta-data.xml` file has been changed the command `ncs-project setup` should be used to bring the project content up to date.

A package, defined in the `project-meta-data.xml` file, can be located at a remote git repository and will then be cloned; or the package may be local to the project itself.

If a package version is specified to origin from a git repository, it may refer to a particular git commit hash, a branch or a tag. This way it is possible to either lock down an exact package version or always make use of the latest version of a particular branch.

A package can also be specified as *local*, which means that it exists in place and no attempts to retrieve it will be made. Note however that it still needs to be a proper package with a `package-meta-data.xml` file.

There is also an option to create a project from an exported bundle. The bundle is generated using the `ncs-project export` command.

OPTIONS

<code>-h, --help</code>	Print a short help text and exit.
<code>-d, --dest Directory</code>	Specify the project (directory) location. The directory will be created if not existing. If not specified, the <i>project-name</i> will be used.
<code>-u, --ncs-bin-url URL</code>	Specify the exact URL pointing to an NCS install binary. Can be a <i>http://</i> or <i>file:///</i> URL.
<code>--from-bundle=<bundle_path> URL</code>	Specify the exact path pointing to a bundled NCS Project. The bundle should have been created using the <i>ncs-project export</i> command.

EXAMPLES

Generate a project using whatever NCS we have in our PATH.

```
$ ncs-project create foo-project
Creating directory: /home/my/foo-project
using locally installed NCS
wrote project to /home/my/foo-project
```

Generate a project using a particular NCS release, located at a particular directory.

```
$ ncs-project create -u file:///lab/releases/ncs-4.0.1.linux.x86_64.installer.bin foo-project
Creating directory: /home/my/foo-project
cp /lab/releases/ncs-4.0.1.linux.x86_64.installer.bin /home/my/foo-project
Installing NCS...
INFO Using temporary directory /tmp/ncs_installer.25681 to stage NCS installation bundle
INFO Unpacked ncs-4.0.1 in /home/my/foo-project/ncs-installdir
INFO Found and unpacked corresponding DOCUMENTATION_PACKAGE
INFO Found and unpacked corresponding EXAMPLE_PACKAGE
INFO Generating default SSH hostkey (this may take some time)
INFO SSH hostkey generated
INFO Environment set-up generated in /home/my/foo-project/ncs-installdir/ncsrc
INFO NCS installation script finished
INFO Found and unpacked corresponding NETSIM_PACKAGE
INFO NCS installation complete

Installing NCS...done
DON'T FORGET TO: source /home/my/foo-project/ncs-installdir/ncsrc
wrote project to /home/my/foo-project
```

Generate a project using a project bundle created with the export command.

```
$ ncs-project create --from-bundle=test_bundle-1.0.tar.gz --dest=installs
Using NCS 4.2.0 found in /home/jvikman/dev/taillf/ncs_dir
wrote project to /home/my/installs/test_bundle-1.0
```

After a project has been created, we need to have its `project-meta-data.xml` file updated before making use of the *ncs-project update* command.

Name

ncs-project-export — Command to create a bundle from a NCS project

Synopsis

```
ncs-project export [OPTIONS] project-name
```

DESCRIPTION

Collects relevant packages and files from an existing NCS project and saves them in a tar file - a *bundle*. This exported bundle can then be distributed to be unpacked, either with the *ncs-project create* command, or simply unpacked using the standard *tar* command.

The bundle is declared in the `project-meta-data.xml` file in the *bundle* section. The packages included in the bundle are leafrefs to the packages defined at the root of the model. We can also define a specific tag, commit or branch, even a different location for the packages, different from the one used while developing. For example we might develop against an experimental branch of a repository, but bundle with a specific release of that same repository. Tags or commit SHA hashes are recommended since branch HEAD pointers usually are a moving target. Should a branch name be used, a warning is issued.

A list of extra files to be included can be specified.

Url references will not be built, i.e they will be added to the bundle as is.

The list of packages to be included in the bundle can be picked from git repositories or locally in the same way as when updating an NCS Project.

Note that the generated `project-meta-data.xml` file, included in the bundle, will specify all the packages as *local* to avoid any dangling pointers to non-accessible git repositories.

OPTIONS

<code>-h, --help</code>	Print a short help text and exit.
<code>-v, --verbose</code>	Print debugging information when creating the bundle.
<code>--prefix=<prefix></code>	Add a prefix to the bundle file name. Cannot be used together with the name option.
<code>--pkg-prefix=<prefix></code>	Use a specific prefix for the compressed packages used in the bundle instead of the default "ncs-\$lt;vsn>" where the <vsn> is the NCS version that ncs-project is shipped with.
<code>--name=<name></code>	Skip any configured name and use <i>name</i> as the bundle file name.
<code>--skip-build</code>	When the packages have been retrieved from their different locations, this option will skip trying to build the packages. No (re-)build will occur of the packages. This can be used to export a bundle for a different NCS version.
<code>--skip-pkg-update</code>	This option will not try to use the package versions defined in the "bundle" part of the project-meta-data, but instead use whatever versions are installed in the "packages" directory. This can be used to export modified packages. Use with care.
<code>--snapshot</code>	Add a timestamp to the bundle file name.

EXAMPLES

Generate a bundle, this command is run in a directory containing a NSO project.

```
$ ncs-project export
Creating bundle ...
Creating bundle ... ok
```

We can also export a bundle with a specific name, below we will create a bundle called `test.tar.gz`.

```
$ ncs-project export --name=test
Creating bundle ...
Creating bundle ... ok
```

Example of how to specify some extra files to be included into the bundle, in the `project-meta-data.xml` file.

```
<bundle>
  <name>test_bundle</name>
  <includes>
    <file>
      <path>README</path>
    </file>
    <file>
      <path>ncs.conf</path>
    </file>
  </includes>
  ...
</bundle>
```

Example of how to specify packages to be included in the bundle, in the `project-meta-data.xml` file.

```
<bundle>
  ...
  <package>
    <name>resource-manager</name>
    <git>
      <repo>ssh://git@stash.tail-f.com/pkg/resource-manager.git</repo>
      <tag>1.2</tag>
    </git>
  </package>
  <!-- Use the repos specified in '../../packages-store' -->
  <package>
    <name>id-allocator</name>
    <git>
      <tag>1.0</tag>
    </git>
  </package>
  <!-- A local package -->
  <!-- (the version will be picked from the package-meta-data.xml) -->
  <package>
    <name>my-local</name>
    <local/>
  </package>
</bundle>
```

Example of how to extract only the packages using `tar`.

```
tar xzf my_bundle-1.0.tar.gz my_bundle-1.0/packages
```

The command uses a temporary directory called `.bundle`, the directory contains copies of the included packages, files and `project-meta-data.xml`. This temporary directory is removed by the export command. Should it remain for some reason it can safely be removed.

The tar-ball can be extracted using `tar` and the packages can be installed like any other packages.

Name

ncs-project-git — For each package git repo, execute a git command

Synopsis

```
ncs-project git [OPTIONS]
```

DESCRIPTION

When developing a project which has many packages coming from remote git repositories, it is convenient to be able to run git commands over all those packages. For example, to display the latest diff or log entry in each and every package. This command makes it possible to do exactly this.

Note that the generated top project Makefile already contain two make targets (gstat and glog) to perform two very common functions for showing any changed but uncommitted files and for showing the last log entry. The same functions can be achieved with this command, although it may require some more typing, see the example below.

OPTIONS

Any git command, including options.

EXAMPLES

Show the latest log entry in each package.

```
$ ncs-project git --no-pager log -n 1

----- Package: esc
commit ccdf889f5fe46d92b5901c7faa9c749f500c68f9
Author: Bill Smith <void@cisco.com>
Date:   Wed Oct 14 10:46:38 2015 +0200

    Getting the latest model changes

----- Package: cisco-ios
commit 05a221ab024108e311709d6491ba8526c31df0ed
Merge: ea72b1e 82e281e
Author: tailf-stash.gen@cisco.com <void@tail-f.com>
Date:   Wed Oct 14 21:09:10 2015 +0200

    Merge pull request #8 in NED/cisco-ios

....
```



Name

`ncs-project-setup` — Command to setup and maintain an NCS project

Synopsis

```
ncs-project setup [OPTIONS] project-name
```

DESCRIPTION

This command is deprecated, please use the *update* command instead.



Name

ncs-project-update — Command to update and maintain an NCS project

Synopsis

ncs-project update [OPTIONS] project-name

DESCRIPTION

Update and maintain an NCS project. This involves fetching packages as defined in `project-meta-data.xml`, and/or update already fetched packages.

For packages, specified to origin from a git repository, a number of git commands will be performed to get it up to date. First a *git stash* will be performed in order to protect from potential data loss of any local changes made. Then a *git fetch* will be made to bring in the latest commits from the origin (remote) git repository. Finally, the local branch, tag or commit hash will be restored, with a *git reset*, according to the specification in the `project-meta-data.xml` file.

Any package specified as *local* will be left unaffected.

Any package which, in its `package-meta-data.xml` file, has a required dependency, will have that dependency resolved. First, if a *packages-store* has been defined in the `project-meta-data.xml` file. The dependant package will be search for in that location. If this fails, an attempt to checkout the dependant package via git will be attempted.

The *ncs-project update* command is intended to be called as soon as you want to bring your project up to date. Each time called, the command will recreate the `setup.mk` include file which is intended to be included by the top Makefile. This file will contain make targets for compiling the packages and to setup any netsim devices.

OPTIONS

-h, --help	Print a short help text and exit.
-v	Print information messages about what is being done.
-y	Answer yes on every questions. Will cause overwriting to any earlier <i>setup.mk</i> files.
--ncs-min-version	
--ncs-min-version-non-strict	
--use-bundle-packages	Update using the packages defined in the bundle section.

EXAMPLES

Bring a project up to date.

```
$ ncs-project update -v
ncs-project: installing packages...
ncs-project: updating package alu-sr...
ncs-project: cd /home/my/mps-vpn-project/packages/alu-sr
ncs-project: git stash # (to save any local changes)
ncs-project: git checkout -q "master"
ncs-project: git fetch
ncs-project: git reset --hard origin/master
ncs-project: updating package alu-sr...done
```

```
ncs-project: installing packages...ok
ncs-project: resolving package dependencies...
ncs-project: filtering missing pkgs for
    "/home/my/mpls-vpn-project/packages/ipaddress-allocator"
ncs-project: missing packages:
[{"<"resource-manager">,>,undefined}]
ncs-project: No version found for dependency: "resource-manager" ,
    trying git and the master branch
ncs-project: git clone "ssh://git@stash.tail-f.com/pkg/resource-manager.git"
    "/home/my/mpls-vpn-project/packages/resource-manager"
ncs-project: git checkout -q "master"
ncs-project: filtering missing pkgs for
    "/home/my/mpls-vpn-project/packages/resource-manager"
ncs-project: missing packages:
[{"<"cisco-ios">,>,<"3.0.2">,>}]
ncs-project: unpacked tar file:
    "/store/releases/ncs-pkgs/cisco-ios/3.0.4/ncs-3.0.4-cisco-ios-3.0.2.tar.gz"
ncs-project: resolving package dependencies...ok
```

Name

`ncs-project` — Command to invoke NCS project commands

Synopsis

`ncs-project` `command` [`OPTIONS`]

DESCRIPTION

This command is used to invoke one of the NCS project commands.

An NCS project is a complete running NCS installation. It can contain all the needed packages and the config data that is required to run the system.

The NCS project is described in a `project-meta-data.xml` file according to the `tailf-ncs-project.yang` Yang model. By using the `ncs-project` commands, the complete project can be populated. This can be used for encapsulating NCS demos or even a full blown turn-key system.

Each command is described in its own man-page, which can be displayed by calling: **`ncs-project help <command>`**

The *OPTIONS* are forwarded to the the invoked script verbatim.

command

<code>create</code>	Create a new NCS project.
<code>export</code>	Export an NCS project.
<code>git</code>	For each git package repository, execute an arbitrary git command.
<code>update</code>	Populate a new NCS project or update an existing project. NOTE: Was called 'setup' earlier.
<code>help command</code>	Display the man-page for the specified NCS project command.



Name

ncs-setup — Command to create an initial NCS setup

Synopsis

```
ncs-setup --dest Directory [--netsim-dir Directory] [--ned-package Dir|Name...] [--generate-ssh-keys] [--use-copy] [--no-netsim]
```

```
ncs-setup --eclipse-setup [--dest Directory]
```

```
ncs-setup --reset [--dest Directory]
```

DESCRIPTION

The **ncs-setup** command is used to create an initial execution environment for a "local install" of NCS. It does so by generating a set of files and directories together with an ncs.conf file. The files and directories are created in the --dest *Directory*, and NCS can be launched in that self-contained directory. For production, it is recommended to instead use a "system install" - see the [NSO Installation Guide](#) and [ncs-installer\(1\)](#).

Without any options an NCS setup without any default packages is created. Using the --netsim-dir and --ned-package options, initial environments for using NCS towards simulated devices, real devices, or a combination thereof can be created.



Note

This command is not included by default in a "system install" of NCS (see [ncs-installer\(1\)](#)), since it is not usable in such an installation. The (single) execution environment is created by the NCS installer when it is invoked with the --system-install option.

OPTIONS

--dest <i>Directory</i>	ncs-setup generates files and directories, all files are written into the --dest directory. The directory is created if non existent.
--netsim-dir <i>Directory</i>	<p>If you have an existing ncs-netsim simulation environment, that environment consists of a set of devices. These devices may be NETCONF, CLI or SNMP devices and the ncs-netsim tool can be used to create, control and manipulate that simulation network.</p> <p>A common developer use case with ncs-setup is that we wish to use NCS to control a simulated network. The option --netsim-dir sets up NCS to manage all the devices in that simulated network. All hosts in the simulated network are assumed to run on the same host as ncs. ncs-setup will generate an XML initialization file for all devices in the simulated network.</p>
--ned-package <i>Directory</i> <i>Name</i>	<p>When you want to create an execution environment where NCS is used to control real, actual managed devices we can use the --ned-package option. The option can be given more than once to add more packages at the same time.</p> <p>The main purpose of this option is to creates symbolic links in ./packages to the NED package(s) indicated to the command. This makes sure that NCS finds the packages when it starts.</p>

For all NED packages that ship together with NCS, i.e packages that are found under `$NCS_DIR/packages/neds` we can just provide the name of the NED. We can also give the path to a NED package.

To setup NCS to manage Juniper and Cisco routers we execute:

```
$ ncs-setup --ned-package juniper --ned-package ios
```

If we have developed our own NED package to control our own ACME router, we can do:

```
$ ncs-setup --ned-package /path/to/acme-package
```

`--generate-ssh-keys`

This option generates fresh ssh keys. By default the keys in `${NCS_DIR}/etc/ncs/ssh` are used. This is useful so that the ssh keys don't change when a new NCS release is installed. Each NCS release comes with newly generated SSH keys.

`--use-copy`

By default, `ncs-setup` will create relative symbolic links in the `./packages` directory. This option copies the packages instead.

`--no-netsim`

By default, `ncs-setup` searches upward in the directory hierarchy for a `netsim` directory. The chosen `netsim` directory will be used to populate the initial CDB data for the managed devices. This option disables this behavior.

Once the initial execution environment is set up, these two options can be used to assist setting up an Eclipse environment or cleaning up an existing environment.

`--eclipse-setup`

When developing the Java code for an NCS application, this command can be used to setup eclipse .project and .classpath appropriately. The .classpath will also contain that source path to all of the NCS Java libraries.

`--reset`

This option resets all data in NCS to "factory defaults" assuming that the layout of the NCS execution environment is created by `ncs-setup`. All CDB database files and all log files are removed. The daemon is also stopped

SIMULATION EXAMPLE

If we have a NETCONF device (which has a set of YANG files and we wish to create a simulation environment for those devices we may combine the three tools 'ncs-make-package', 'ncs-netsim' and 'ncs-setup' to achieve this. Assume all the yang files for the device resides in `/path/to/yang` we need to

- Create a package for the YANG files.

```
$ ncs-make-package --netconf-ned /path/to/yang acme
```

This creates a package in `./acme`

- Setup a network simulation environment. We choose to create a simulation network with 5 routers named `r0` to `r4` with the `ncs-netsim` tool.

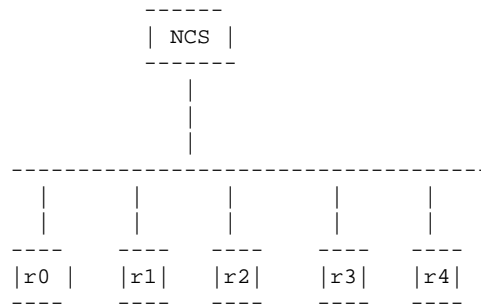
```
$ ncs-netsim create-network ./acme 5 r
```

The network simulation environment will be created in `./netsim`

- Finally create a directory where we execute NCS

```
$ ncs-setup --netsim-dir netsim --dest ./acme_nms \  
--generate-ssh-keys  
$ cd ./acme_nms; ncs-setup --eclipse-setup
```

This results in a simulation environment that looks like:



with NCS managing 5 simulated NETCONF routers, all running ConfD on localhost (on different ports) and all running the YANG models from `/path/to/yang`



Name

ncs-uninstall — Command to remove NCS installation

Synopsis

```
ncs-uninstall --ncs-version [Version] [--install-dir InstallDir] [--non-interactive]
```

```
ncs-uninstall --all [--install-dir InstallDir] [--non-interactive]
```

DESCRIPTION

The **ncs-uninstall** command can be used to remove part or all of an NCS "system installation", i.e. one that was done with the `--system-install` option to the NCS installer (see [ncs-installer\(1\)](#)).

OPTIONS

<code>--ncs-version [<i>Version</i>]</code>	Removes the installation of static files for NCS version <i>Version</i> . I.e. the directory tree rooted at <i>InstallDir/ncs-Version</i> will be removed. The <i>Version</i> argument may also be given as the filename or pathname of the installation directory, or, unless <code>--non-interactive</code> is given, omitted completely in which case the command will offer selection from the installed versions.
<code>--all</code>	Completely removes the NCS installation. I.e. the whole directory tree rooted at <i>InstallDir</i> , as well as the directories for config files (option <code>--config-dir</code> to the installer), run-time state files (option <code>--run-dir</code> to the installer), and log files (option <code>--log-dir</code> to the installer), and also the init script and user profile scripts.
<code>[--install-dir <i>InstallDir</i>]</code>	Specifies the directory for installation of NCS static files, like the <code>--install-dir</code> option to the installer. If this option is omitted, <code>/opt/ncs</code> will be used for <i>InstallDir</i> .
<code>[--non-interactive]</code>	If this option is used, removal will proceed without asking for confirmation.



Name

ncs — command to start and control the NCS daemon

Synopsis

```
ncs [--conf ConfFile] [--cd Dir] [--addloadpath Dir] [--nolog ] [--smp Nr] [ --foreground [ -v  
| --verbose ] [--stop-on-eof ] [--with-package-reload ] [--ignore-initial-validation ] [--full-upgrade-  
validation ] [--start-phase0 ] [ --epoll { true | false } ]  
  
ncs { --wait-phase0[ TryTime] | --start-phase1 | --start-phase2 | --wait-started[ TryTime] | --reload | --  
areload | --status | --check-callbacks [ Namespace | Path] | --loadfile File | --rollback Nr | --debug-  
dump File | --cli-j-dump File | --loadxmlfiles File... | --mergexmlfiles File... | --stop } [--timeout  
MaxTime]  
  
ncs { --version | --cdb-debug-dump Directory }
```

DESCRIPTION

Use this command to start and control the NCS daemon.

STARTING NCS

These options are relevant when starting the NCS daemon.

<code>-c, --conf <i>ConfFile</i></code>	<i>ConfFile</i> is the path to a <i>ncs.conf</i> file. If the <code>-c <i>File</i></code> argument is not given to <i>ncs</i> , first <code>\$PWD</code> is searched for a file called <i>ncs.conf</i> , if not found <code>\$NCS_DIR/etc/ncs/ncs.conf</code> is chosen.
<code>--cd <i>Dir</i></code>	Change working directory
<code>--addloadpath <i>Dir</i></code>	Add <i>Dir</i> to the set of directories NCS uses to load fxs, clispec, NCS packages and SNMP bin files.
<code>--nolog</code>	Do not log initial startup messages to syslog.
<code>--smp <i>Nr</i></code>	Number of threads to run for Symmetric Multiprocessing (SMP). The default is to enable SMP support, with as many threads as the system has logical processors, if more than one logical processor is detected. Giving a value of 1 will disable SMP support, while a value bigger than 1 will enable SMP support, where NCS will at any given time use at most as many logical processors as the number of threads.
<code>--foreground [-v --verbose] [--stop-on-eof]</code>	Do not start as a daemon. Can be used to start NCS from a process manager. In combination with <code>-v</code> or <code>--verbose</code> , all log messages are printed to stdout. Useful during development. In combination with <code>--stop-on-eof</code> , NCS will stop if it receives EOF (ctrl-d) on standard input. Note that to stop NCS when run in foreground, send EOF (if <code>--stop-on-eof</code> was used) or use <code>ncs --stop</code> . Do not terminate with ctrl-c, since NCS in that case won't have the chance to close the database files.
<code>--with-package-reload</code>	When NCS starts, if the private package directory tree already exists, NCS will load the packages from this directory tree and not search the load-path for packages. If the <code>--with-package-reload</code> option is given when starting NCS, the load-path will be searched and the packages found there copied to the private package directory

`--ignore-initial-validation`

`--full-upgrade-validation`

`--start-phase0`

`--epoll { true | false }`

tree, replacing the previous contents, before loading. This should always be used when upgrading to a new version of NCS in an existing directory structure, to make sure that new packages are loaded together with the other parts of the new system.

When NCS is started from the `/etc/init.d` scripts, that get generated by the `--system-install` option to the NCS installer, the environment variable `NCS_RELOAD_PACKAGES` can be set to 'true' to attempt a package reload.

When CDB starts on an empty database, or when upgrading, it starts a transaction to load the initial configuration or perform the upgrade. This option makes NCS skip any validation callpoints when committing these initial transaction. (The preferred alternative is to use start-phases and register the validation callpoints in phase 0, see the user guide).

Perform a full validation of the entire database if the data models have been upgraded. This is useful in order to trigger external validation to run even if the database content has not been modified.

Start the daemon, but only start internal subsystems and CDB. Phase 0 is used when a controlled upgrade is done.

Determines whether NCS should use an enhanced `poll()` function (e.g. Linux `epoll(7)`). This can improve performance when NCS has a high number of connections, but there may be issues with the implementation in some OS/kernel versions. The default is true.

COMMUNICATING WITH NCS

When the NCS daemon has been started, these options are used to communicate with the running daemon.

By default these options will perform their function by connecting to a running NCS daemon over the loopback interface on the standard port. If the environment variable `NCS_IPC_PORT` and/or `NCS_IPC_ADDR` are set then the address/port in those variables will be used to communicate with the NCS daemon. (Must be used if the daemon is not listening on its standard port on localhost, see the `/ncs-config/ncs-ipc-address/ip` setting in the [ncs.conf\(5\)](#) man-page, and the section on [NCS IPC](#) in the NCS Users Guide).

`--wait-phase0`
`[TryTime]`

This call hangs until NCS has initialized start phase0. After this call has returned, it is safe to register validation callbacks, upgrade CDB etc. This function is useful when NCS has been started with `--foreground` and `--start-phase0`. It will keep trying the initial connection to NCS for at most TryTime seconds (default 5).

`--start-phase1`

Do not start the subsystems that listen to the management IP address. Must be called after the daemon was started with `--start-phase0`.

`--start-phase2`

Must be called after the management interface has been brought up, if `--start-phase1` has been used. Starts the subsystems that listens to the management IP address.

`--wait-started`
`[TryTime]`

This call hangs until NCS is completely started. This function is useful when NCS has been started with `--foreground`. It will keep trying the initial connection to NCS for at most TryTime seconds (default 5).

<code>--reload</code>	Reload the NCS daemon configuration. All log files are closed and reopened, which means that ncs --reload can be used from e.g. <code>logrotate(8)</code> , but it is more efficient to use ncs_cmd -c reopen_logs for this purpose. Note: If we update a <code>.fxs</code> file it is not enough to do a reload; the "packages reload" action must be invoked, or the daemon must be restarted with the <code>--with-package-reload</code> option.
<code>--areload</code>	Asynchronously reload the NCS daemon configuration. This can be used in scripts executed by the NCS daemon.
<code>--stop</code>	Stop the NCS daemon.
<code>--status</code>	Prints status information about the NCS daemon on stdout. Among the things listed are: loaded namespaces, current user sessions, callpoints (and whether they are registered or not), CDB status, and the current start-phase. Start phases are reported as "status:" and can be one of starting (which is pre-phase0), phase0, phase1, started (i.e. phase2), or stopping (which means that NCS is about to shutdown).
<code>--debug-dump File</code>	Dump debug information from an already running NCS daemon into a file. The file only makes sense to NCS developers. It is often a good idea to include a debug dump in NCS trouble reports.
<code>--cli-j-dump File</code>	Dump cli structure information from the NCS daemon into a file.
<code>--check-callbacks [Namespace Path]</code>	Walks through the entire data tree (config and stat), or only the Namespace or Path, and verifies that all read-callbacks are implemented for all elements, and verifies their return values.
<code>--loadfile File</code>	Load configuration in curly bracket format from File.
<code>--rollback Nr</code>	Rollback configuration to saved configuration number Nr.
<code>--loadxmlfiles File ...</code>	Load configuration in XML format from Files. The configuration is completely replaced by the contents in Files.
<code>--mergexmlfiles File ...</code>	Load configuration in XML format from Files. The configuration is merged with the contents in Files. The XML may use the 'operation' attribute, in the same way as it is used in a NETCONF <code><edit-config></code> operation.
<code>--timeout MaxTime</code>	Specify the maximum time to wait for the NCS daemon to complete the command, in seconds. If this option is not given, no timeout is used..

STANDALONE OPTIONS

<code>--cdb-debug-dump Directory</code>	Print a lot of information about the CDB files in <i>Directory</i> to stdout. This is a completely stand-alone feature and the only thing needed is the <code>.cdb</code> files (no running NCS daemon or <code>.fxs</code> files etc).
<code>--version</code>	Reports the ncs version without interacting with the daemon.
<code>--timeout MaxTime</code>	See above

ENVIRONMENT

When NCS is started from the `/etc/init.d` scripts, that get generated by the `--system-install` option to the NCS installer, the environment variable `NCS_RELOAD_PACKAGES` can be set to 'true' to attempt a package reload.

DIAGNOSTICS

If NCS starts, the exit status is 0. If not it is a positive integer. The different meanings of the different exit codes are documented in the "NCS System Management" chapter in the user guide. When failing to start, the reason is stated in the NCS daemon log. The location of the daemon log is specified in the ConfFile as described in [ncs.conf\(5\)](#).

SEE ALSO

[ncs.conf\(5\)](#) - NCS daemon configuration file format

Name

ncs_cli — Frontend to the NSO CLI engine

Synopsis

```
ncs_cli [options] [File]
```

```
ncs_cli [ --help ] [ --host Host ] [ --ip IpAddress | IpAddress/Port ] [ --address Address ] [ --port PortNumber ] [ --cwd Directory ] [ --proto tcp> | ssh | console ] [ --interactive ] [ --noninteractive ] [ --user Username ] [ --uid UidInt ] [ --groups Groups ] [ --gids GidList ] [ --gid Gid ] [ --opaque Opaque ] [ --noaaa ]
```

DESCRIPTION

The ncs_cli program is a C frontend to the NSO CLI engine. The **ncs_cli** program connects to NSO and basically passes data back and forth from the user to NSO.

ncs_cli can be invoked from the command line. If so, no authentication is done. The archetypical usage of ncs_cli is to use it as a login shell in /etc/passwd, in which case authentication is done by the login program.

OPTIONS

-h, --help	Display help text.
-H, --host <i>HostName</i>	Gives the name of the current host. The ncs_cli program will use the value of the system call <code>gethostbyname()</code> by default. The host name is used in the CLI prompt.
-i, --ip <i>IpAddress</i> <i>IpAddress/Port</i>	Set the IP (or IP address and port) which NSO reports that the user is coming from. The ncs_cli program by default tries to determine this automatically by reading the <code>SSH_CONNECTION</code> environment variable.
-A, --address <i>Address</i>	CLI address to connect to. The default is 127.0.0.1. This can be controlled by either this flag, or the UNIX environment variable <code>NCS_IPC_ADDR</code> . The <code>-A</code> flag takes precedence.
-P, --port <i>PortNumber</i>	CLI port to connect to. The default is the NSO IPC port, which is 4569. This can be controlled by either this flag, or the UNIX environment variable <code>NCS_IPC_PORT</code> . The <code>-P</code> flag takes precedence.
-c, --cwd <i>Directory</i>	The current working directory for the user once in the CLI. All file references from the CLI will be relative to the cwd. By default the value will be the actual cwd where ncs_cli is invoked.
-p, --proto <i>ssh tcp console</i>	The protocol the user is using. If <code>SSH_CONNECTION</code> is set, this defaults to "ssh", otherwise "console".
-n, --interactive	This forces the CLI to run in interactive mode. In non interactive mode, the CLI never prompts the user for any input. This flag can sometimes be useful in certain CLI scripting scenarios.
-N, --noninteractive	This forces the CLI to run in non interactive mode. See ??? for further info.
-u, --user <i>User</i>	Indicates to NSO which username the user has. This defaults to the username of the invoker.
-U, --uid <i>Uid</i>	Indicates to NSO which uid the user has.

<code>-g, --groups <i>GroupList</i></code>	Indicates to NSO which groups the user are a member of. The parameter is a comma separated string. This defaults to the actual UNIX groups the user is a member of. The group names are used by the AAA system in NSO to authorize data and command access.
<code>-D, --gids <i>GidList</i></code>	Indicates to NSO which secondary group ids the user shall have. The parameter is a comma separated string of integers. This defaults to the actual secondary UNIX group ids the user has. The gids are used by NSO when NSO executes commands on behalf of the user.
<code>-G, --gid <i>Gid</i></code>	Indicates to NSO which group id the user shall have. This defaults to the actual UNIX group id the user has. The gid is used by NSO when NSO executes commands on behalf of the user.
<code>-O, --opaque <i>Opaque</i></code>	Pass an opaque string to NCS. The string is not interpreted by NCS, only made available to application code. See "built-in variables" in clispec(5) and <code>maapi_get_user_session_opaque()</code> in confd_lib_maapi(3) . The string can be given either via this flag, or via the UNIX environment variable <code>NCS_CLI_OPAQUE</code> . The <code>-O</code> flag takes precedence.
<code>--noaaa</code>	Completely disables all AAA checks for this CLI. This can be used as a disaster recovery mechanism if the AAA rules in NSO have somehow become corrupted.

ENVIRONMENT VARIABLES

<code>NCS_IPC_ADDR</code>	Which IP address to connect to.
<code>NCS_IPC_PORT</code>	Which TCP port to connect to.
<code>SSH_CONNECTION</code>	Set by openssh and used by <code>ncs_cli</code> to determine client IP address etc.
<code>TERM</code>	Passed on to terminal aware programs invoked by NSO.

EXIT CODES

- 0 Normal exit
- 1 Failed to read user data for initial handshake.
- 2 Close timeout, client side closed, session inactive.
- 3 Idle timeout triggered.
- 4 Tcp level error detected on daemon side.
- 5 Internal error occurred in daemon.
- 5 User interrupted clistart using special escape char.
- 6 User interrupted clistart using special escape char.
- 7 Daemon abruptly closed socket.

SCRIPTING

It is very easy to use **ncs_cli** from `/bin/sh` scripts. **ncs_cli** reads stdin and can then also be run in non interactive mode. This is the default if stdin is not a tty (as reported by `isatty()`)

Here is example of invoking **ncs_cli** from a shell script.

```
#!/bin/sh

ncs_cli << EOF
```

```
configure
set foo bar 13
set funky stuff 44
commit
exit no-confirm
exit
EOF
```

And here is an example capturing the output of **ncs_cli**:

```
#!/bin/sh
{ ncs_cli << EOF;
configure
set trap-manager t2 ip-address 10.0.0.1 port 162 snmp-version 2
commit
exit no-confirm
exit
EOF
} | grep 'Aborted:.*not unique.*'
if [ $? != 0 ]; then
    echo 'test2: commit did not fail'; exit 1;
fi
```

The above type of CLI scripting is a very efficient and easy way to test various aspects of the CLI.



Name

`ncs_cmd` — Command line utility that interfaces to common NSO library functions

Synopsis

```
ncs_cmd [(1) options] [filename]

ncs_cmd [(1) options] -c string

ncs_cmd -h | -h commands | -h command-name...
(1) [-r | -o | -e | -S ] [-f [w] | [p] [ r | s ] ] [-a address] [-p port] [-u user] [-g group] [-x context]
[-s] [-m] [-h] [-d]
```

DESCRIPTION

The **ncs_cmd** utility is implemented as a wrapper around many common CDB and MAAPI function calls. The purpose is to make it easier to prototype and test various NSO issues using normal scripting tools.

Input is provided as a file (default `stdin` unless a filename is given) or as directly on the command line using the `-c string` option. The **ncs_cmd** expects commands separated by semicolon (;) or newlines. A pound (#) sign means that the rest of the line is treated as a comment. For example:

```
ncs_cmd -c get_phase
```

Would print the current start-phase of NSO, and:

```
ncs_cmd -c "get_phase ; get_txid"
```

would first print the current start-phase, then the current transaction ID of CDB.

Sessions towards CDB, and transactions towards MAAPI are created as-needed. At the end of the script any open CDB sessions are closed, and any MAAPI read/write transactions are committed.

OPTIONS

`-d` Debug flag. Add more to increase debug level. All debug output will be to `stderr`.
`-m` Don't load the schemas at startup.

ENVIRONMENT VARIABLES

`NCS_IPC_ADDR` The address used to connect to the NSO daemon, overrides the compiled in default.
`NCS_IPC_PORT` The port number to connect to the NSO daemon on, overrides the compiled in default.

EXAMPLES

- 1 Getting the address of `eth0`

```
ncs_cmd -c "get /sys:sys/ifc{eth0}/ip"
```

- 2 Setting a leaf in CDB operational

```
ncs_cmd -o -c "set /sys:sys/ifc{eth0}/stat/tx 88"
```

- 3 Making NSO running on localhost the master, with the name `node0`

```
ncs_cmd -c "master node0"
```

Then tell the NSO also running on localhost, but listening on port 4566, slave to the master. Calling the slave node1

```
ncs_cmd -p 4566 -c "slave node1 node0 127.0.0.1"
```


Name

`ncs_load` — Command line utility to load and save NSO configurations

Synopsis

```
ncs_load [-W] [-S] [(I) common options] [filename]
```

```
ncs_load -l [-m | -r | -j | -n ] [-D] [(I) common options] [filename...]
```

```
ncs_load -C [-R] [filename...]
```

```
ncs_load -h | -?
```

```
(1) [-d] [-t] [-F { x | p | o | j | c | i } ] [ -H | -U ] [-a] [-e] [ [-u user] [-g group...] [-c context] | [-i]] [[-p  
keypath] | [-P XPath]] [-o] [-s] [-O] [-M]
```

DESCRIPTION

This command provides a convenient way of loading and saving all or parts of the configuration in different formats. It can be used to initialize or restore configurations as well as in CLI commands.

If you run **ncs_load** without any options it will print the current configuration in XML format on stdout. The exit status will be zero on success and non-zero otherwise.

COMMON OPTIONS

<code>-d</code>	Debug flag. Add more to increase debug level. All debug output will be to stderr.
<code>-t</code>	Measure how long the requested command takes and print the result on stderr.
<code>-F x p o j c i</code>	Selects the format of the configuration, must be set both when loading and saving. One of XML (x), pretty XML (p), JSON (o), curly braces J-style CLI (j), C-style CLI (c), or I-style CLI (i). Default is XML.
<code>-H</code>	Hide all hidden nodes. By default, no nodes are hidden unless ncs_load has attached to an existing transaction, in which case the hidden nodes are the same as in that transaction's session.
<code>-U</code>	Unhide all hidden nodes. By default, no nodes are hidden unless ncs_load has attached to an existing transaction, in which case the hidden nodes are the same as in that transaction's session.
<code>-u user, -g group ..., -c context</code>	Loading and saving the configuration is done in a user session, using these options it is possible to specify which user, groups (more than one <code>-g</code> can be used to add groups), and context that should be used when starting the user session. If only a user is supplied the user is assumed to belong to a single group with the same name as the user. This is significant in that AAA rules will be applied for the specified user / groups / context combination. The default is to use the <code>system</code> context, which implies that AAA rules will <i>not</i> be applied at all.

**Note**

If the environment variables `NCS_MAAPI_USID` and `NCS_MAAPI_THANDLE` are set (see the ENVIRONMENT section), or if the `-i` option is used, these options are silently ignored, since **ncs_load** will attach to an existing transaction.

`-i`

Instead of starting a new user session and transaction, **ncs_load** will try to attach to the init session. This is only valid when NSO is in start phase 0, and will fail otherwise. It can be used to load a “factory default” file during startup, or loading a file during upgrade.

SAVE CONFIGURATION

By default the complete current configuration will be output on stdout. To save it in a file add the filename on the command line (the `-f` option is deprecated). The file is opened by the **ncs_load** utility, permissions and ownership will be determined by the user running **ncs_load**. Output format is specified using the `-F` option.

When saving the configuration in XML format, the context of the user session (see the `-c` option) will determine which namespaces with export restriction (from `tailf:export`) that are included. If the `system` context is used (this is the default), all namespaces are saved, regardless of export restriction. When saving the configuration in one of the CLI formats, the context used for this selection is always `cli`.

A number of options are only applicable, or have a special meaning when saving the configuration:

- `-f filename` Filename to save configuration to (option is deprecated, just give the filename on the command line).
- `-W` Include leaves which are unset (set to their default value) in the output. By default these leaves are not included in the output.
- `-S` Include the default value of a leaf as a comment (only works for CLI formats, not XML). (Corresponds to the `MAAPI_CONFIG_SHOW_DEFAULTS` flag).
- `-p keypath` Only include the configuration below *keypath* in the output.
- `-P XPath` Filter the configuration using the *XPath* expression. (Only works for the XML format.)
- `-o` Include operational data in the output. (Corresponds to the `MAAPI_CONFIG_WITH_OPER` flag).
- `-O` Include *only* operational data, and ancestors to operational data nodes, in the output. (Corresponds to the `MAAPI_CONFIG_OPER_ONLY` flag).
- `-M` Include NCS service-meta-data attributes in the output. (Corresponds to the `MAAPI_CONFIG_WITH_SERVICE_META` flag).

LOAD CONFIGURATION

When the `-l` option is present **ncs_load** will load all the files listed on the command line. The file(s) are expected to be in XML format unless otherwise specified using the `-F` flag. Note that it is the NSO daemon that opens the file(s), it must have permission to do so. However relative pathnames are assumed to be relative to the working directory of the **ncs_load** command.

If neither of the `-m` and `-r` options are given when multiple files are listed on the command line, **ncs_load** will silently treat the second and subsequent files as if `-m` had been given, i.e. it will merge in the contents of these files instead of deleting and replacing the configuration for each file. Note, we almost always want the merge behavior. If no file is given, or `-` is given as a filename, **ncs_load** will stream standard input to NSO.

<code>-f filename</code>	The file to load (deprecated, just list the file after the options instead).
<code>-m</code>	Merge in the contents of <i>filename</i> , the (somewhat unfortunate) default is to delete and replace.
<code>-j</code>	Do not run FASTMAP, if FASTMAPPED service data is loaded, we sometimes do not want to run the mapper code. One example is a backup saved in XML format that contains both device data and also service data.
<code>-n</code>	Only load data to CDB inside NCS, do not attempt to perform any update operations towards the managed devices. This corresponds to the 'no-networking' flag to the commit command in the NCS CLI.
<code>-x</code>	Lax loading. Only applies to XML loading. Ignore unknown namespaces, attributes and elements.
<code>-r</code>	Replace the part of the configuration that is present in <i>filename</i> , the default is to delete and replace. (Corresponds to the <code>MAAPI_CONFIG_REPLACE</code> flag).
<code>-a</code>	When loading configuration in 'i' or 'c' format, do a commit operation after each line. Default and recommended is to only commit when all the configuration has been loaded. (Corresponds to the <code>MAAPI_CONFIG_AUTOCOMMIT</code> flag).
<code>-e</code>	When loading configuration do not abort when encountering errors (corresponds to the <code>MAAPI_CONFIG_CONTINUE_ON_ERROR</code> flag).
<code>-D</code>	Delete entire config before loading.
<code>-p keypath</code>	Delete everything below <i>keypath</i> before loading the file.
<code>-o</code>	Accept but ignore contents in the file which is operational data (without this flag it will be an error).
<code>-O</code>	Start an operational transaction and load operational data (via a transaction instead of via CDB). (Only allowed for XML format.)

LOAD CDB OPERATIONAL

The `-C` option is used to load operational data. When you use `-C` all other options except `-R` (and `-d`) are ignored, since they don't apply. Files on the command line must be in XML format. If no file is given, or `"-"` is given as a filename, **ncs_load** will read standard input. If the `-R` option is included, CDB operational subscription notifications will be generated.

Any data which isn't part of CDB operational per the data model will be ignored. This means that you can save a single file with both configuration and operational data and feed it back to **ncs_load**.

If you use a relative path for *filename* it is assumed to be relative to the working directory of the **ncs_load** command

EXAMPLES

Example 1. Reloading all xml files in the cdb directory

```
ncs_load -D -m -l cdb/*.xml
```

Example 2. Merging in the contents of `conf.cli`

```
ncs_load -l -m -F j conf.cli
```

Example 3. Print interface config and statistics data in cli format

```
ncs_load -F i -o -p /sys:sys/ifc
```

Example 4. Using xslt to format output

```
ncs_load -F x -p /sys:sys/ifc | xsltproc fmtifc.xsl -
```

Example 5. Using xmllint to pretty print the xml output

```
ncs_load -F x | xmllint --format -
```

Example 6. Saving config and operational data to /tmp/conf.xml

```
ncs_load -F x -o > /tmp/conf.xml
```

Example 7. Restoring both config and operational data

```
ncs_load -l -F x -o /tmp/conf.xml  
ncs_load -C /tmp/conf.xml
```

Example 8. Measure how long it takes to fetch config

```
ncs_load -t > /dev/null  
elapsed time: 0.011 s
```

Example 9. Output all instances in list /foo/table which has ix larger than 10

```
ncs_load -F x -P "/foo/table[ix > 10]"
```

ENVIRONMENT

NCS_IPC_ADDR	The address used to connect to the NSO daemon, overrides the compiled in default.
NCS_IPC_PORT	The port number to connect to the NSO daemon on, overrides the compiled in default.
NCS_MAAPI_USID, NCS_MAAPI_THANDLE	If set ncs_load will attach to an existing transaction in an existing user session instead of starting a new session.

These environment variables are set by the NSO CLI when it invokes external commands, which means you can run **ncs_load** directly from the CLI. For example, the following addition to the `<operationalMode>` in a clispec file (see [clispec\(5\)](#))

```
<cmd name="servers" mount="show">  
  <info/>  
  <help/>  
  <callback>  
    <exec>  
      <osCommand>ncs_load</osCommand>  
      <args>-F j -p /system/servers</args>  
    </exec>  
  </callback>  
</cmd>
```

will add a **show servers** command which, when run will invoke **ncs_load -F j -p /system/servers**. This will output the configuration below `/system/servers` in curly braces format.

Note that when these environment variables are set, it means that the configuration will be loaded into the current CLI transaction (which must be in configure mode, and have AAA permissions to actually modify the config).

To load (or save) a file in a separate transaction, unset these two environment variables before invoking the **ncs_load** command.



Name

ncsc — NCS Yang compiler

Synopsis

```
ncsc -c [-a | --annotate YangAnnotationFile] [--deviation DeviationFile] [-o FxsFile]
[--verbose] [--fail-on-warnings] [-E | --error ErrorCode...] [-W | --warning ErrorCode...] [-w | --
no-warning ErrorCode...] [--strict-yang] [--no-yang-source] [--use-description [always]] [--no-
features] | [-F | --features Features ...] [--ignore-unknown-features] [-p | --prefix Prefix] [--yangpath
YangDir] [--export Agent [-f FxsFileOrDir...] ...] -- YangFile
```

```
ncsc --strip-yang-source FxsFile
```

```
ncsc --list-errors
```

```
ncsc -c [-o CclFile] ClispecFile
```

```
ncsc -c [-o BinFile] [-I Dir] MibFile
```

```
ncsc -c [-o BinFile] [--read-only] [--verbose] [-I Dir] [--include-file BinFile] [--fail-on-warnings]
[--warn-on-type-errors] [--warn-on-access-mismatch] [--mib-annotation MibA] [-f FxsFileOrDir...] --
MibFile FxsFile...
```

```
ncsc --ncs-compile-bundle Directory [--yangpath YangDir] [--ncs-skip-template] [--ncs-skip-
statistics] [--ncs-skip-config] [--lax-revision-merge] [--ncs-depend-package PackDir] [--ncs-apply-
deviations] --ncs-device-type netconf | snmp-ned | generic-ned | cli-ned --ncs-device-
dir Directory
```

```
ncsc --ncs-compile-mib-bundle Directory [--ncs-skip-template] [--ncs-skip-statistics] [--ncs-skip-
config] --ncs-device-type netconf | snmp-ned | generic-ned | cli-ned --ncs-device-dir
Directory
```

```
ncsc --ncs-compile-module YangFile [--yangpath YangDir] [--ncs-skip-template] [--ncs-skip-
statistics] [--ncs-skip-config] [--ncs-keep-callpoints] [--lax-revision-merge] [--ncs-depend-package
PackDir] --ncs-device-type netconf | snmp-ned | generic-ned | cli-ned --ncs-device-
dir Directory
```

```
ncsc --emit-java JFile [--print-java-filename] [--java-disable-prefix] [--java-package Package] [--
exclude-enums] [--fail-on-warnings] [-f FxsFileOrDir ...] FxsFile
```

```
ncsc --emit-python PyFile [--print-python-filename] [--no-init-py] [--python-disable-prefix] [--
exclude-enums] [--fail-on-warnings] [-f FxsFileOrDir ...] FxsFile
```

```
ncsc --emit-mib MibFile [--join-names capitalize | hyphen] [--oid OID] [--top Name] [--
tagpath Path] [--import Module Name] [--module Module] [--generate-oids] [--generate-yang-
annotation] [--skip-symlinks] [--top Top] [--fail-on-warnings] [--no-comments] [--read-only] --
FxsFile...
```

```
ncsc --mib2yang-std [-p Prefix] [-o YangFile] -- MibFile
```

```
ncsc --mib2yang-mods [--mib-annotation MibA] [--keep-readonly] [--namespace Uri] [--revision Date]
[-o YangDeviationFile] -- MibFile
```

```
ncsc --mib2yang [--mib-annotation MibA] [--emit-doc] [--snmp-name] [--read-only] [-u Uri] [-p
Prefix] [-o YangFile] -- MibFile
```

```

ncsc --snmpuser EngineID User AuthType PrivType PassPhrase

ncsc --revision-merge [-o ResultFxs] [-v ] [-f FxsFileOrDir...] -- ListOfFxsFiles...

ncsc --lax-revision-merge [-o ResultFxs] [-v ] [-f FxsFileOrDir...] -- ListOfFxsFiles...

ncsc --get-info FxsFile

ncsc --get-uri FxsFile

ncsc --version

```

DESCRIPTION

During startup the NSO daemon loads .fxs files describing our configuration data models. A .fxs file is the result of a compiled YANG data model file. The daemon also loads clispec files describing customizations to the auto-generated CLI. The clispec files are described in [clispec\(5\)](#).

A yang file by convention uses .yang (or .yin) filename suffix. YANG files are directly transformed into .fxs files by ncsc.

We can use any number of .fxs files when working with the NSO daemon.

The --emit-java option is used to generate a .java file from a .fxs file. The java file is used in combination with the Java library for Java based applications.

The --emit-python option is used to generate a .py file from a .fxs file. The python file is used in combination with the Python library for Python based applications.

The --print-java-filename option is used to print the resulting name of the would be generated .java file.

The --print-python-filename option is used to print the resulting name of the would be generated .py file.

The --python-disable-prefix option is used to prevent prepending the Yang module prefix to each symbol in the generated .py file.

A clispec file by convention uses a .cli filename suffix. We use the ncsc command to compile a clispec into a loadable format (with a .ccl suffix).

A mib file by convention uses a .mib filename suffix. The ncsc command is used for compiling the mib with one or more fxs files (containing OID to YANG mappings) into a loadable format (with a .bin suffix). See the NSO User Guide for more information about compiling the mib.

Take a look at the EXAMPLE section for a crash course.

OPTIONS

Common options

<code>-f, --fxsdep <i>FxsFileOrDir...</i></code>	.fxs files (or directories containing .fxs files) to be used to resolve cross namespace dependencies.
<code>--yangpath <i>YangModuleDir</i></code>	YangModuleDir is a directory containing other YANG modules and submodules. This flag must be used when we import or

Compile options

<code>-o, --output <i>File</i></code>	include other YANG modules or submodules that reside in another directory. Put the resulting file in the location given by <i>File</i> .
<code>-c, --compile <i>File</i></code>	Compile a YANG file (.yang/.yin) to a .fxs file or a clispec (.cli file) to a .ccl file, or a MIB (.mib file) to a .bin file
<code>-a, --annotate <i>AnnotationFile</i></code>	YANG users that are utilizing the tailf:annotate extension must use this flag to indicate the YANG annotation file(s). This parameter can be given multiple times.
<code>--deviation <i>DeviationFile</i></code>	Indicates that deviations from the module in <i>DeviationFile</i> should be present in the fxs file. This parameter can be given multiple times.
<code>-F<i>features</i>, --feature <i>features</i></code>	Indicates that support for the YANG <i>features</i> should be present in the fxs file. <i>features</i> is a string on the form <i>modulename</i> : <i>[feature(,feature)*]</i> This option is used to prune the data model by removing all nodes that are defined with a "if-feature" that is not listed as <i>feature</i> . This option can be given multiple times. If this option is not given, nothing is pruned, i.e., it works as if all features were explicitly listed. If the module uses a feature defined in an imported YANG module, it must be given as <i>modulename:feature</i> . By default, the YANG module and submodules source is included in the fxs file, so that a NETCONF or RESTCONF client can download the module from the server. If this option is given, the YANG source is not included.
<code>--no-yang-source</code>	Indicates that no YANG features from the given module are supported.
<code>--no-features</code>	Instructs the compiler to not give an error if an unknown feature is specified with <code>--feature</code> .
<code>--ignore-unknown-features</code>	NCS needs to have a unique prefix for each loaded YANG module, which is used e.g. in the CLI and in the APIs. By default the prefix defined in the YANG module is used, but this prefix is not required to be unique across modules. This option can be used to specify an alternate prefix in case of conflicts. The special value 'module-name' means that the module name will be used for this prefix.
<code>-p, --prefix <i>Prefix</i></code>	Normally, 'description' statements are ignored by ncsc. Instead the 'tailf:info' statement is used as help and information text in the CLI and Web UI. When this option is specified, text in 'description' statements is used if no 'tailf:info' statement is present. If the option <i>always</i> is given, 'description' is used even if 'tailf:info' is present.
<code>--use-description [<i>always</i>]</code>	Makes the namespace visible to Agent. Agent is either "none", "all", "netconf", "snmp", "cli", "webui", "rest" or a free-text string. This option overrides any <code>tailf:export</code> statements in the module. The option "all" makes it visible to all agents. Use "none" to make it invisible to all agents.
<code>--export Agent ...</code>	

<code>--fail-on-warnings</code>	Make compilation fail on warnings.
<code>-W <i>ErrorCode</i></code>	Treat <i>ErrorCode</i> as a warning, even if <code>--fail-on-warnings</code> is given. <i>ErrorCode</i> must be a warning or a minor error. Use <code>--list-errors</code> to get a listing of all errors and warnings.
	The following example treats all warnings except the warning for dependency mismatch as errors:
<code>-w <i>ErrorCode</i></code>	<pre>\$ ncsc -c --fail-on-warnings -W TAILF_DEPENDENCY_MISMATCH</pre> Do not report the warning <i>ErrorCode</i> , even if <code>--fail-on-warnings</code> is given. <i>ErrorCode</i> must be a warning. Use <code>--list-errors</code> to get a listing of all errors and warnings.
	The following example ignores the warning TAILF_DEPENDENCY_MISMATCH:
<code>-E <i>ErrorCode</i></code>	<pre>\$ ncsc -c -w TAILF_DEPENDENCY_MISMATCH</pre> Treat the warning <i>ErrorCode</i> as an error. Use <code>--list-errors</code> to get a listing of all errors and warnings.
	The following example treats only the warning for unused import as an error:
<code>--strict-yang</code>	<pre>\$ ncsc -c -E UNUSED_IMPORT</pre> Force strict YANG compliance. Currently this checks that the <code>deref()</code> function is not used in XPath expressions and leafrefs.

Standard MIB to YANG options

<code>--mib2yang-std <i>MibFile</i></code>	Generate a YANG file from the MIB module (<i>.mib</i> file), in accordance with the IETF standard, RFC-6643. If the MIB IMPORTs other MIBs, these MIBs must be available (as <i>.mib</i> files) to the compiler when a YANG module is generated. By default, all MIBs in the current directory and all builtin MIBs are available. Since the compiler uses the tool smidump to perform the conversion to YANG, the environment variable <code>SMIPATH</code> can be set to a colon-separated list of directories to search for MIB files.
<code>-p, --prefix <i>Prefix</i></code>	Specify a prefix to use in the generated YANG module. An appendix to the RFC describes how the prefix is automatically generated, but such an automatically generated prefix is not always unique, and NSO requires unique prefixes in all loaded modules.

Standard MIB to YANG modification options

<code>--mib2yang-mods <i>MibFile</i></code>	Generate a combined YANG deviation/annotation file from the MIB module (<i>.mib</i> file), which can be used to compile the yang file generated by <code>--mib2yang-std</code> , to achieve a similar result as with the non-standard <code>--mib2yang</code> translation.
<code>-p, --prefix <i>Prefix</i></code>	Specify a prefix to use in the generated YANG module. An appendix to the RFC describes how the prefix is automatically generated, but such an automatically generated prefix is not always unique, and NSO requires unique prefixes in all loaded modules.

<code>--mib-annotation <i>MibA</i></code>	Provide a MIB annotation file to control how to override the standard translation of specific MIB objects to YANG. See mib_annotations(5) .
<code>--revision <i>Date</i></code>	Generate a revision statement with the provided <i>Date</i> as value in the deviation/annotation file.
<code>--namespace <i>Uri</i></code>	Specify a uri to use as namespace in the generated deviation/annotation module.
<code>--keep-readonly</code>	Do not generate any deviations of the standard config (false) statements. Without this flag, config statements will be deviated to true on yang nodes corresponding to writable MIB objects.

MIB to YANG options

<code>--mib2yang <i>MibFile</i></code>	Generate a YANG file from the MIB module (.mib file). If the MIB IMPORTs other MIBs, these MIBs must be available (as .mib files) to the compiler when a YANG module is generated. By default, all MIBs in the current directory and all builtin MIBs are available. Since the compiler uses the tool smidump to perform the conversion to YANG, the environment variable SMIPATH can be set to a colon-separated list of directories to search for MIB files.
<code>-u, --uri <i>Uri</i></code>	Specify a uri to use as namespace in the generated YANG module.
<code>-p, --prefix <i>Prefix</i></code>	Specify a prefix to use in the generated YANG module.
<code>--mib-annotation <i>MibA</i></code>	Provide a MIB annotation file to control how to translate specific MIB objects to YANG. See mib_annotations(5) .
<code>--snmp-name</code>	Generate the YANG statement "tailf:snmp-name" instead of "tailf:snmp-oid".
<code>--read-only</code>	Generate a YANG module where all nodes are "config false".

MIB compiler options

<code>-c, --compile <i>MibFile</i></code>	Compile a MIB module (.mib file) to a .bin file. If the MIB IMPORTs other MIBs, these MIBs must be available (as compiled .bin files) to the compiler. By default, all compiled MIBs in the current directory and all builtin MIBs are available. Use the parameters <code>--include-dir</code> or <code>--include-file</code> to specify where the compiler can find the compiled MIBs.
<code>--verbose</code>	Print extra debug info during compilation.
<code>--read-only</code>	Compile the MIB as read-only. All SET attempts over SNMP will be rejected.
<code>-I, --include-dir <i>Dir</i></code>	Add the directory <i>Dir</i> to the list of directories to be searched for IMPORTed MIBs (.bin files).
<code>--include-file <i>File</i></code>	Add <i>File</i> to the list of files of IMPORTed (compiled) MIB files. File must be a .bin file.
<code>--fail-on-warnings</code>	Make compilation fail on warnings.
<code>--warn-on-type-errors</code>	Warn rather than give error on type checks performed by the MIB compiler.
<code>--warn-on-access-mismatch</code>	Give a warning if an SNMP object has read only access to a config object.

`--mib-annotation MibA`

Provide a MIB annotation file to fine-tune how specific MIB objects should behave in the SNMP agent. See [mib_annotations\(5\)](#).

Emit SMIv2 MIB options

`--emit-mib MibFile`

Generates a MIB file for use with SNMP agents/managers. See the appropriate section in the SNMP agent chapter in the NSO User Guide for more information.

`--join-names capitalize`

Join element names without separator, but capitalizing, to get the MIB name. This is the default.

`--join-names hyphen`

Join element names with hyphens to get the MIB name.

`--join-names force-
capitalize`

The characters '.' and '_' can occur in YANG identifiers but not in SNMP identifiers; they are converted to hyphens, unless this option is given. In this case, such identifiers are capitalized (to lowerCamelCase).

`--oid OID`

Let *OID* be the top object's OID. If the first component of the OID is a name not defined in SNMPv2-SMI, the `--import` option is also needed in order to produce a valid MIB module, to import the name from the proper module. If this option is not given, a `tailf:snmp-oid` statement must be specified in the YANG header.

`--tagpath Path`

Generate the MIB only for a subtree of the module. The *Path* argument is an absolute schema node identifier, and it must refer to container nodes only.

`--import Module Name`

Add an IMPORT statement which imports *Name* from the MIB *Module*.

`--top Name`

Let *Name* be the name of the top object.

`--module Name`

Let *Name* be the module name. If a `tailf:snmp-mib-module-name` statement is in the YANG header, the two names must be equal.

`--generate-oids`

Translate all data nodes into MIB objects, and generate OIDs for data nodes without `tailf:snmp-oid` statements.

`--generate-yang-
annotation`

Generate a YANG annotation file containing the `tailf:snmp-oid`, `tailf:snmp-mib-module-name` and `tailf:snmp-row-status-column` statements for the nodes. Implies `--skip-symlinks`.

`--skip-symlinks`

Do not generate MIB objects for data nodes modeled through symlinks.

`--fail-on-warnings`

If this option is used all warnings are treated as errors and ncsc will fail its execution.

`--no-comments`

If this option is used no additional comments will be generated in the MIB.

`--read-only`

If this option is used all objects in the MIB will be read only.

`--prefix String`

Prefix all MIB object names with *String*.

Emit SNMP user options

`--snmpuser EngineID
User AuthType PrivType
PassPhrase`

Generates a user entry with localized keys for the specified engine identifier. The output is an `usmUserEntry` in XML format that can be used in an initiation file for the SNMP-USER-BASED-

SM-MIB::usmUserTable. In short this command provides key generation for users in SNMP v3. This option takes five arguments: The EngineID is either a string or a colon separated hexlist, or a dot separated octet list. The User argument is a string specifying the user name. The AuthType argument is one of md5, sha, or none. The PrivType argument is one of des, aes, or none. The PassPhrase argument is a string.

Emit Java options

<code>--emit-java <i>JFile</i></code>	Generate a .java ConfNamespace file from a .fxs file to be used when working with the Java library. The file is useful, but not necessary when working with the NAVU library. JFile could either be a file or a directory. If JFile is a directory the resulting .java file will be created in that directory with a name based on the module name in the YANG module. If JFile is not a directory that file is created. Use <code>--print-java-filename</code> to get the resulting file name.
<code>--print-java-filename</code>	Only print the resulting java file name. Due to restrictions of identifiers in Java the name of the Class and thus the name of the file might get changed if non Java characters are used in the name of the file or in the name of the module. If this option is used no file is emitted the name of the file which would be created is just printed on stdout.
<code>--java-package <i>Package</i></code>	If this option is used the generated java file will have the given package declaration at the top.
<code>--exclude-enums</code>	If this option is used, definitions for enums are omitted from the generated java file. This can in some cases be useful to avoid conflicts between enum symbols, or between enums and other symbols.
<code>--fail-on-warnings</code>	If this option is used all warnings are treated as errors and ncsc will fail its execution.
<code>-f, --fxsdep <i>FxsFileOrDir...</i></code>	.fxs files (or directories containing .fxs files) to be used to resolve cross namespace dependencies.

NCS device module import options

These options are used to import device modules into NCS. The import is done as a source code transformation of the yang modules (MIBs) that define the managed device. By default, the imported modules (MIBs) will be augmented three times. Once under **/devices/device/config**, once under **/devices/template/config** and once under **/devices/device/live-status**.

The **ncsc** commands to import device modules can take the following options:

- `--ncs-skip-template` This option makes the NCS bundle compilation skip the layout of the template tree - thus making the NCS feature of provisioning devices through the template tree unusable. The main reason for using this option is to save memory if the data models are very large.
- `--ncs-skip-statistics` This option makes the NCS bundle compilation skip the layout of the live tree. This option make sense for e.g NED modules that are sometimes config only. It also makes sense for the Junos module which doesn't have and "config false" data.
- `--ncs-skip-config` This option makes the NCS bundle compilation skip the layout of the config tree. This option make sense for some NED modules that are typically status and commands only.

`--ncs-keep-callpoints` This option makes the NCS bundle compilation keep callpoints when performing the ncs transformation from modules to device modules, as long as the callpoints have either `tailf:set-hook` or `tailf:transaction-hook` as sub statement.

`--ncs-device-dir` *Directory* This is the target directory where the output of the `--ncs-compile-xxx` command is collected.

`--lax-revision-merge` When we have multiple revisions of the same module, the `ncsc` command to import the module will fail if a YANG module does not follow the YANG module upgrade rules. See RFC 6020. This option makes `ncsc` ignore those strict rules. Use with extreme care, the end result may be that NCS is incompatible with the managed devices.

`--ncs-depend-package` *PackageDir* When a package has references to Yang code in another package, use this flag when compiling the package.

`--ncs-apply-deviations` This option will make `--ncs-compile-bundle` apply deviations that are defined in one module with a target in another module. By default such deviations are ignored.

`--ncs-device-type` *netconf | snmp-ned | generic-ned | cli-ned* All imported device modules adhere to a specific device type.

`--ncs-compile-bundle`
YangFileDirectory

To import a set of managed device YANG files into NCS, gather the required files in a directory and import by using this flag. Several invocations will populate the mandatory `--ncs-device-dir` directory with the compiler output. This command also handles revision management for NCS imported device modules. Invoke the command several times with different *YangFileDirectory* directories and the same `--ncs-device-dir` directory to accumulate the revision history of the modules in several different *YangFileDirectory* directories.

This command will ignore any NCS-internal modules found in the *YangFileDirectory* directory, as well as modules annotating, deviating, or augmenting NCS-internal modules. For the remaining ones, modules having annotations or deviations for other modules are identified, and such annotations and deviations are processed as follows:

- 1 Annotations using `tailf:annotate` are ignored (this annotation mechanism is incompatible with the source code transformation).
- 2 Annotations using `tailf:annotate-module` are applied (but may, depending on the type of annotation and the device type, be ignored by the transformation).
- 3 Deviations are applied if the `--ncs-apply-deviations` option is given, otherwise ignored. The deviations are not applied by default, since NCS does not support different devices having different deviations for a common YANG module.

Typically when NCS needs to manage multiple revisions of the same module, the filenames of the YANG modules are on the form of `MOD@REVISION.yang`. The `--ncs-compile-bundle` as well as the `--ncs-compile-module` commands will rename the source YANG files and organize the result as per revision in the `--ncs-device-dir` output directory.

The output structure could look like:

```
ncsc-out
|---modules
|----|----fxs
|----|----|----interfaces.fxs
|----|----|----sys.fxs
|----|----revisions
|----|----|----interfaces
|----|----|----|----revision-merge
|----|----|----|----|----interfaces.fxs
|----|----|----|----|----2009-12-06
|----|----|----|----|----|----interfaces.fxs
|----|----|----|----|----|----interfaces.yang.orig
|----|----|----|----|----|----interfaces.yang
|----|----|----|----|----2006-11-05
|----|----|----|----|----|----interfaces.fxs
|----|----|----|----|----|----interfaces.yang.orig
|----|----|----|----|----|----interfaces.yang
|----|----|----sys
|----|----|----|----2010-03-26
|----|----|----|----|----sys.yang.orig
|----|----|----|----|----sys.yang
|----|----|----|----|----sys.fxs
|----|----yang
|----|----|----interfaces.yang
|----|----|----sys.yang
```

where we have the following paths:

- 1 modules/fxs contains the FXS files that are revision compiled and are ready to load into NCS.
- 2 modules/yang/\$MODULE.yang is the augmented YANG file of the latest revision. NCS will run with latest revision of all YANG files, and the revision compilation will annotate that tree with information indication at which revision each YANG element was introduced.
- 3 modules/revisions/\$MODULE contains the different revisions for \$MODULE and also the merged compilation result.

--ncs-compile-mib-bundle
MibFileDirectory

To import a set of SNMP MIB modules for a managed device into NCS, put the required MIBs in a directory and import by using this flag. The MIB files MUST have the ".mib" extension. The compile also picks up any MIB annotation files present in this directory, with the extension ".miba". See [mib_annotations\(5\)](#).

This command translates all MIB modules to YANG modules according to the standard translation algorithm defined in I.D-ietf-netmod-smi-yang, then it generates a YANG deviations module in order to handle writable configuration data. When all MIB modules have been translated to YANG, --ncs-compile-bundle is invoked.

Each invocation of this command will populate the --ncs-device-dir with the compiler output. This command also handles revision management for NCS imported device modules. Invoke the command several times with different *MibFileDirectory* directories and the same --ncs-device-dir directory to accumulate the revision

history of the modules in several different *MibFileDirectory* directories.

`--ncs-compile-module
YangFile`

This ncsc command imports a single device YANG file into the `--ncs-model-dir` structure. It's an alternative to `--ncs-compile-bundle`, however is just special case of a one-module bundle. From a Makefile perspective it may sometimes be easier to use this version of bundle compilation.

Misc options

`--strip-yang-source
FxsFile`

Removes included YANG source from the fxs file. This makes the file smaller, but it means that the YANG module and submodules cannot be downloaded from the server, unless they are present in the load path.

`--get-info FxsFile`

Various info about the file is printed on standard output, including the names of the source files used to produce this file, which ncsc version was used, and for fxs files, namespace URI, other namespaces the file depends on, namespace prefix, and mount point.

`--get-uri FxsFile`

Extract the namespace URI.

`--version`

Reports the ncsc version.

EXAMPLE

Assume we have the file `system.yang`:

```
module system {
  namespace "http://example.com/ns/gargleblaster";
  prefix "gb";

  import ietf-inet-types {
    prefix inet;
  }
  container servers {
    list server {
      key name;
      leaf name {
        type string;
      }
      leaf ip {
        type inet:ip-address;
      }
      leaf port {
        type inet:port-number;
      }
    }
  }
}
```

To compile this file we do:

```
$ ncsc -c system.yang
```

If we intend to manipulate this data from our Java programs, we must typically also invoke:

```
$ ncsc --emit-java blaster.java system.fxs
```

Finally we show how to compile a clispec into a loadable format:


```
$ ncsc -c mycli.cli
$ ls mycli.ccl
myccl.ccl
```

DIAGNOSTICS

On success exit status is 0. On failure 1. Any error message is printed to stderr.

YANG 1.1

NCS supports YANG 1.1, as defined in draft-ietf-netmod-rfc6020bis-12, with the following exceptions:

- Type `empty` in unions and in list keys is not supported.
- Type `leafref` in unions are not validated, and treated as a string internally.
- **anydata** is not supported.
- The new scoping rules for submodules are not implemented. Specifically, a submodule must still include other submodules in order to access definitions defined there.
- Inline **notification** statements are handled by the compiler, but not supported in the APIs.
- The new XPath functions `derived-from()` and `derived-from-or-self()` can only be using with literal strings in the second argument.
- Leafref paths without prefixes in top-level typedefs are handled as in YANG 1.

SEE ALSO

The NCS User Guide

`ncs(1)`

`ncs.conf(5)`

`clispec(5)`

`mib_annotations(5)`

command to start and control the NCS daemon

NCS daemon configuration file format

CLI specification file format

MIB annotations file format



Name

nct-backup — perform an NCS backup on Host(s)

Synopsis

nct-backup [*OPTIONS*]

DESCRIPTION

The nct-backup(1) will perform an NCS backup on the specified Host(s) using the built in NCS functionality. At success, the name of the created backup file will be returned.

OPTIONS

-c, --cmd <i>CMD</i>	The <i>CMD</i> parameter can be any of: ‘backup’, ‘list’, ‘revert-latest’ or ‘restore’, where ‘backup’ is the default command. To perform an NCS backup use ‘backup’. To list the available backup files use ‘list’. To revert the latest backup file use ‘revert-latest’. To restore a specific backup use ‘restore’ with the ‘--file’ switch.
--file <i>FILE</i>	File name of the NCS backup file. Example: ‘ncs-4.1@2016-01-15T12:58:14.backup’.
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir</i> /ncs-VSN, allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir</i> /current pointing to one of the ncs-VSN directories. If the --install-dir option is omitted, /opt/ncs will be used for <i>InstallDir</i> .
--run-dir <i>RunDir</i>	This is the directory where runtime specific files are store. Specifically, for this command, it is under this directory that backup files, created with the ncs-backup command, is stored.

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the --group switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.

--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user, "bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user, "bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups, ["service", "london"]} , ... ] }.
{ "192.168.24.98", [ {groups, ["service", "paris"]} , ... ] }.
{ "192.168.23.11", [ {groups, ["device", "london"]} , ... ] }.
{ "192.168.24.12", [ {groups, ["device", "paris"]} , ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations

only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{ "192.168.23.99", [ { name, "pariss"}, ... ] }.
{ "192.168.23.98", [ { name, "londons"}, ... ] }.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...
nct stop --name londons --hostsfile ...
```



Name

nct-check — check the availability of a cluster of nodes

Synopsis

nct-check [*OPTIONS*]

DESCRIPTION

The nct-check(1) is a command for checking that the Host access over SSH is ok, that various NCS API's are running and if HA is configured.

OPTIONS

-c, --cmd <i>CMD</i>	The type of check to be executed is controlled by this switch. The default is <i>all</i> , i.e to do all the available checks. Other types are: <i>ssh</i> (check SSH access is ok), <i>ssh-sudo</i> (check that SSH and SUDO is possible), 'disk-usage' (return some disk usage information), <i>rest</i> (is the REST API accessible), <i>netconf</i> (is the NETCONF API accessible), <i>ncs-vsn</i> (what NCS version is the REST API announcing), <i>ha</i> (is HA configured and if so how).
--disk-limit <i>DISKLIMIT</i>	The 'disk-usage' check will attach a <i>WARNING</i> label in the output if the disk usage is above <i>DISKLIMIT</i> . Default is 80 percent.
--netconf-pass <i>NETCONFPASS</i>	The NETCONF password to be used (default: admin).
--netconf-port <i>NETCONFPORT</i>	The NETCONF Port number to be used (default: 2022).
--netconf-proto <i>NETCONFPORT</i>	The NETCONF bearer protocol to be used (default: ssh). Currently, only <i>ssh</i> is supported.
--netconf-user <i>NETCONFUSER</i>	The NETCONF user to be used (default: admin).
--rest-pass <i>RESTPASS</i>	The REST password to be used.
--rest-port <i>RESTPORT</i>	The REST Port number to be used (default: 8888).
--rest-user <i>RESTUSER</i>	The REST user to be used (default: \$USER).
--rest-ssl <i>BOOLEAN</i>	REST via SSL true/false (default: true).
--ssh-cmd <i>SSHCMD</i>	The <i>SSHCMD</i> argument is a string that can contain any shell command, or sequence of shell commands, that you want to execute on the host(s). Note that the command(s) are executed as the <i>SSHUSER</i> so you may have to run the commands with <i>sudo</i> for root preveleges and such. Example: "sudo mv /tmp/foo /etc/bar". Tips: if it is a long running command you may want to use the <i>nohup</i> command on the host(s). See the <i>nohup</i> man-page.
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir/ncs-VSN</i> , allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir/current</i> pointing to one of the <i>ncs-VSN</i> directories. If the <i>--install-dir</i> option is omitted, <i>/opt/ncs</i> will be used for <i>InstallDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS } .  
{ "192.168.23.99", OPTIONS } .  
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:


```

{"192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{"192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...

```

The *OPTIONS* is enclosed with `[]` brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the `'-'` in a switch corresponds to a `'_'` in the hostsfile (e.g `--ssh-user` vs `ssh_user`). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```

{"192.168.23.99",[ {groups,[ "service","london"]}, ... ]}.
{"192.168.24.98",[ {groups,[ "service","paris"]}, ... ]}.
{"192.168.23.11",[ {groups,[ "device","london"]}, ... ]}.
{"192.168.24.12",[ {groups,[ "device","paris"]}, ... ]}.

```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achived by using the `'--group'` switch to a NCS tools command. For example:

```

nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...

```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the `'--groupoper and'` switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified (`'--groupoper or'`).

SSH

It is possible to specify the `'SSH User'` and `'SSH Password'` to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the `'SSH Password'` to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use `'SSH KEYS'` as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the `'ssh-copy-id'` command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the `--ssh-key-name` switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{"192.168.23.99",[{name, "pariss"}, ... ]}.  
{"192.168.23.98",[{name, "londons"}, ... ]}.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```

Name

nct-cli-cmd — execute a CLI command on host(s)

Synopsis

nct-cli-cmd [*OPTIONS*]

DESCRIPTION

The nct-cli-cmd(1) is a command to execute CLI commands to one or several host(s).

OPTIONS

-c, --cmd <i>CMD</i>	The <i>CMD</i> argument is a string that can contain any valid CLI command that you want to execute on the host(s). Note that the command(s) are piped to the ncs_cli executable.
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir</i> /ncs-VSN, allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir</i> /current pointing to one of the ncs-VSN directories. If the --install-dir option is omitted, /opt/ncs will be used for <i>InstallDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the --group switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you

--ssh-pass <i>SSHPASS</i>	can specify another key name using this switch. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH password to be used. See also the SSH section .
--ssh-timeout <i>SSHTIMEOUT</i>	The SSH Port number to be used (default: 22).
	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user, "bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user, "bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups, ["service", "london"]} , ... ] }.
{ "192.168.24.98", [ {groups, ["service", "paris"]} , ... ] }.
{ "192.168.23.11", [ {groups, ["device", "london"]} , ... ] }.
{ "192.168.24.12", [ {groups, ["device", "paris"]} , ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the

specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified (`--groupoper` or `'`).

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the `--ssh-key-name` switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{ "192.168.23.99", [{name, "pariss"}, ... ] }.  
{ "192.168.23.98", [{name, "londons"}, ... ] }.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-copy — copy a file to the specified host(s)

Synopsis

nct-copy [*OPTIONS*]

DESCRIPTION

The nct-copy(1) will copy a file to a directory on host(s). If no target directory is specified, */tmp* will be used. See the ‘--file’ and ‘--to-dir’ for more details.

OPTIONS

--file <i>FILE</i>	The <i>FILE</i> will be copied to the ‘/tmp’ directory on host(s) unless the ‘--to-dir’ switch specify some other directory.
--to-dir <i>TODIR</i>	The file will be copied to <i>TODIR</i> . Note that the file is copied with the permissions of the <i>SSHUSER</i> . So to achieve the wanted result, in some cases you may be better off copying a file to <i>/tmp</i> and then move it with the right privileges using the <i>nct-check(1)</i> command.

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable ‘NCT_HOSTSFILE’ to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example ‘id_rsa’ for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .

--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{ "192.168.23.99", [ { name, "pariss"}, ... ] }.  
{ "192.168.23.98", [ { name, "londons"}, ... ] }.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-get-logs — Get a tar-ball of the NSO log dir from the specified host(s)

Synopsis

nct-get-logs [*OPTIONS*]

DESCRIPTION

The command `nct-get-logs(1)` will create a tar-ball, with the name `${hostname}-${timestamp}.tar`, of the NSO log dir and copy the tar-ball to the local host from the specified host(s). If no local directory is specified, `/tmp` will be used. If no log-dir is specified, `/var/log/ncs` will be used.

OPTIONS

<code>--local-dir</code> <i>Dir</i>	The directory used to store the fetched tar-balls. If the <code>--local-dir</code> option is omitted, <code>/tmp</code> will be used for <i>Dir</i> .
<code>--log-dir</code> <i>LogDir</i>	This directory is used for the different log files written by NCS. If the <code>--log-dir</code> option is omitted, <code>/var/log/ncs</code> will be used for <i>LogDir</i> .

COMMON OPTIONS

<code>--concurrent</code> <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
<code>--name</code> <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
<code>--group</code> <i>GROUP</i> [<i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
<code>--groupoper</code> <i>GROUPOPER</i>	If several groups are specified with the <code>--group</code> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
<code>--help</code>	Will print out a short help text and then terminate the command.
<code>-h, --host</code> <i>HOST</i> [<i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
<code>--hostsfile</code> <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
<code>--progress</code> <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
<code>--ssh-key-name</code> <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
<code>--ssh-pass</code> <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .

--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{ "192.168.23.99", [ { name, "pariss"}, ... ] }.  
{ "192.168.23.98", [ { name, "londons"}, ... ] }.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-ha — HA handling using the tailf-hcc package

Synopsis

nct-ha [*OPTIONS*]

DESCRIPTION

The nct-ha(1) is a command for invoking tailf-hcc package actions in a cluster environment

OPTIONS

--action *ACTION*

Specify which action to perform on the host(s). Valid actions are:

activate | deactivate | role-override | role-revert | force-be-
status | node-connect-state | cluster-activate
cluster-deactivate | cluster-status | cluster-status
cluster-connect-state | cluster-role-revert
cluster-role-override | readonly | reactivate

The actions are documented in tailf-hcc-actions.yang following the tailf-hcc package. See the [TAILF-HCC section](#).

--role *ROLE*

Specify a HA-role for the node(s). Must be used together with the role-override and cluster-role-override flag. Valid roles are:

unknown | none | slave | master | relay

--mode *BOOLEAN*

readonly for the node(s). Must be used together with the readonly flag. Valid modes are:

true | false

--member *MEMBER*

Specify which member should act as master. Must be used together with the force-be-slave-to flag

--rest-pass *RESTPASS*

The REST password to be used.

--rest-port *RESTPORT*

The REST Port number to be used (default: 8888).

--rest-user *RESTUSER*

The REST user to be used (default: \$USER).

--rest-ssl *BOOLEAN*

REST via SSL true/false (default: true).

COMMON OPTIONS

--concurrent *BOOL*

Each *HOST* will be operated upon concurrently. The default value is *true*. If set to *false*, each *HOST* will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one *HOST* at a time.

--name *NAME*

Restrict the command to only operate on the given *NAME*. See the [NAME section](#).

--group *GROUP* [,*GROUP*]

Restrict the command to only operate on the given *GROUP* or groups. See the [GROUPS section](#).

--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST[,HOST]+</i>	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:


```
{ "192.168.23.99", [ {groups, [ "service", "london" ]}, ... ] }.
{ "192.168.24.98", [ {groups, [ "service", "paris" ]}, ... ] }.
{ "192.168.23.11", [ {groups, [ "device", "london" ]}, ... ] }.
{ "192.168.24.12", [ {groups, [ "device", "paris" ]}, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{ "192.168.23.99", [ {name, "pariss"}, ... ] }.
{ "192.168.23.98", [ {name, "londons"}, ... ] }.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...
nct stop --name londons --hostsfile ...
```

TAILF-HCC

The NCS HA package *tailf-hcc* must be applied to all nodes in master and slave clusters.



Name

nct-hostsfile — a complete list of all available options

Introduction

A hostsfile is a plain text file that consists of host-entries, containing individual options for each entry. Each entry - called a *Tuple*, that begin with a '{' bracket and ends with a '}' - consists of a *Hostname/IP-Address*, enclosed in double quotes and a list of options, where the list begin with a '[' bracket and ends with a corresponding ']' bracket:

```
{ "192.168.23.99", [OPTIONS] }.  
{ "192.168.23.98", [OPTIONS] }.  
...etc...
```

OPTIONS

The options list is a comma separated list of key/value(s) tuples. The available options are:

name	<p>A name representation of the host. Example:</p> <pre>{ "192.168.23.99", [{name, "pariss"}, ...]}. { "192.168.23.98", [{name, "londons"}, ...]}.</pre>
groups	<p>Mandatory for nct-move-device. A list of groups the host belongs to. Example:</p> <pre>{ "192.168.23.99", [{groups, ["groupA", "groupB"]}, ...]}. { "192.168.23.98", [{groups, ["groupB", "groupC"]}, ...]}.</pre>
ssh_user	<p>SSH User for the host. Example:</p> <pre>{ "192.168.23.99", [{ssh_user, "admin"}, ...]}. { "192.168.23.98", [{ssh_user, "oper"}, ...]}.</pre>
ssh_pass	<p>SSH Password for the host. Example:</p> <pre>{ "192.168.23.99", [{ssh_pass, "secret"}, ...]}. { "192.168.23.98", [{ssh_pass, "secret"}, ...]}.</pre>
ssh_port	<p>SSH PORT for the host. Example:</p> <pre>{ "192.168.23.99", [{ssh_port, 22}, ...]}. { "192.168.23.98", [{ssh_port, 24}, ...]}.</pre>
netconf_user	<p>NETCONF User for the host. Example:</p> <pre>{ "192.168.23.99", [{netconf_user, "admin"}, ...]}. { "192.168.23.98", [{netconf_user, "oper"}, ...]}.</pre>
netconf_pass	<p>NETCONF Password for the host. Example:</p> <pre>{ "192.168.23.99", [{netconf_pass, "secret"}, ...]}. { "192.168.23.98", [{netconf_pass, "secret"}, ...]}.</pre>
netconf_port	<p>NETCONF PORT for the host. Example:</p> <pre>{ "192.168.23.99", [{netconf_port, 2022}, ...]}. { "192.168.23.98", [{netconf_port, 2024}, ...]}.</pre>
rest_user	<p>REST User for the host. Example:</p> <pre>{ "192.168.23.99", [{rest_user, "admin"}, ...]}. { "192.168.23.98", [{rest_user, "oper"}, ...]}.</pre>
rest_pass	<p>REST Password for the host. Example:</p>

	<pre>{ "192.168.23.99", [{rest_pass, "secret"}, ...]}. { "192.168.23.98", [{rest_pass, "secret"}, ...]}.</pre>
rest_port	REST PORT for the host. Example: <pre>{ "192.168.23.99", [{rest_port, 8080}, ...]}. { "192.168.23.98", [{rest_port, 8888}, ...]}.</pre>
rest_ssl	REST via ssl. Example: <pre>{ "192.168.23.99", [{rest_ssl, "true"}, ...]}. { "192.168.23.98", [{rest_ssl, "false"}, ...]}.</pre>
install_dir	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir/ncs-VSN</i> , allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir/current</i> pointing to one of the ncs-VSN directories. If the <i>install_dir</i> option is omitted, <i>/opt/ncs</i> will be used. Example: <pre>{ "192.168.23.99", [{install_dir, "/opt/ncs"}, ...]}.</pre>
config_dir	This directory is used for config files, e.g. <i>ncs.conf</i> . If the <i>config_dir</i> option is omitted, <i>/etc/ncs</i> will be used. Example: <pre>{ "192.168.23.99", [{config_dir, "/etc/ncs"}, ...]}.</pre>
run_dir	This directory is used for run-time state files, such as the CDB data base and currently used packages. If the <i>run_dir</i> option is omitted, <i>/var/opt/ncs</i> will be used. Example: <pre>{ "192.168.23.99", [{run_dir, "/var/opt/ncs"}, ...]}.</pre>
log_dir	This directory is used for the different log files written by NCS. If the <i>log_dir</i> option is omitted, <i>/var/log/ncs</i> will be used. Example: <pre>{ "192.168.23.99", [{log_dir, "/var/log/ncs"}, ...]}.</pre>
service_node	The name of the node serving as a service node to the host. Must correspond with the name option of the service node. Example: <pre>{ "192.168.23.99", [{name, "pariss"}, ...]}. { "192.168.23.11", [{service_node, "pariss"}, ...]}.</pre>
device_nodes	Mandatory for nct-move-device. A list of names of the nodes serving as a device nodes to the host. Must correspond with the name option of the device node. Example: <pre>{ "192.168.23.99", [{device_nodes, ["parissd1", "parisd2"]}, ...]} { "192.168.23.11", [{name, "parisd1"}, ...]}. { "192.168.23.12", [{name, "parisd2"}, ...]}.</pre>
	Mandatory for nct-move-device.

Name

nct-install — install an NCS release on host(s)

Synopsis

nct-install [*OPTIONS*]

DESCRIPTION

The nct-install(1) command will install an NCS release on the specified host(s) using the ‘--cmd’ and ‘--file’ switches. If the NCS release already is installed on host(s), the command will fail.

OPTIONS

-c, --cmd <i>CMD</i>	The <i>CMD</i> can be either <i>install</i> or <i>check</i> . The former is to be used together with the ‘--file’ switch to install a particular NCS release on the specified host(s). The output from the NCS install script on each host will be stored in a log file. This log file can be inspected by the latter command (<i>check</i>). Tips: since an install may take some time, you may have to repeat the <i>check</i> command until the output log ends with ‘NCS installation complete’.
--file <i>FILE</i>	The <i>FILE</i> is the name of the NCS install bin-file. Example: ‘ncs-3.3.linux.x86_64.installer.bin’
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir</i> /ncs-VSN, allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir</i> /current pointing to one of the ncs-VSN directories. If the --install-dir option is omitted, /opt/ncs will be used for <i>InstallDir</i> .
--config-dir <i>ConfigDir</i>	This directory is used for config files, e.g. ncs.conf. If the --config-dir option is omitted, /etc/ncs will be used for <i>ConfigDir</i> .
--run-dir <i>RunDir</i>	This directory is used for run-time state files, such as the CDB data base and currently used packages. If the --run-dir option is omitted, /var/opt/ncs will be used for <i>RunDir</i> .
--log-dir <i>LogDir</i>	This directory is used for the different log files written by NCS. If the --log-dir option is omitted, /var/log/ncs will be used for <i>LogDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .

--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g *--ssh-user* vs *ssh_user*). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the *hostsfile* when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups, [ "service", "london" ]}, ... ] }.
{ "192.168.24.98", [ {groups, [ "service", "paris" ]}, ... ] }.
{ "192.168.23.11", [ {groups, [ "device", "london" ]}, ... ] }.
{ "192.168.24.12", [ {groups, [ "device", "paris" ]}, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each *Host*, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the `--ssh-key-name` switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{ "192.168.23.99", [ {name, "pariss"}, ... ] }.
{ "192.168.23.98", [ {name, "londons"}, ... ] }.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...
nct stop --name londons --hostsfile ...
```



Name

nct-load-config — load NCS configuration on host(s)

Synopsis

nct-load-config [*OPTIONS*]

DESCRIPTION

With the `nct-load-config(1)` command it is possible to load xml configuration files to NCS hosts, either running NCS instances or as initial configuration.

OPTIONS

--file <i>FILE</i>	The configuration xml file containing CDB configuration, or an <code>ncs.conf</code> file. Example: <code>`config.xml ncs.conf'</code>
--type <i>TYPE</i>	The type of configuration file. See the TYPE section . Valid types are: <code>cdb-init ncs-conf xml json c-cli j-cli i-cli</code>
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir/ncs-VSN</i> , allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir/current</i> pointing to one of the <code>ncs-VSN</code> directories. If the <code>--install-dir</code> option is omitted, <code>/opt/ncs</code> will be used for <i>InstallDir</i> .
--config-dir <i>ConfigDir</i>	This directory is used for config files, e.g. <code>ncs.conf</code> . If the <code>--config-dir</code> option is omitted, <code>/etc/ncs</code> will be used for <i>ConfigDir</i> .
--run-dir <i>RunDir</i>	This directory is used for run-time state files, such as the CDB data base and currently used packages. If the <code>--run-dir</code> option is omitted, <code>/var/opt/ncs</code> will be used for <i>RunDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .

--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST[,HOST]+</i>	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups, [ "service", "london" ]}, ... ] }.
{ "192.168.24.98", [ {groups, [ "service", "paris" ]}, ... ] }.
{ "192.168.23.11", [ {groups, [ "device", "london" ]}, ... ] }.
{ "192.168.24.12", [ {groups, [ "device", "paris" ]}, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{ "192.168.23.99", [ {name, "pariss"}, ... ] }.
{ "192.168.23.98", [ {name, "londons"}, ... ] }.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...
nct stop --name londons --hostsfile ...
```

TYPE

You can specify which type of configuration file you which to load on the host. The valid types are:

'cddb-init': Specifies that the file contains initial CDB

configuration which will be placed in the NCS cdb-directory.

'ncs-conf': Specifies that the file is an ncs.conf file which will be placed under /etc/ncs.

'xml': Specifies that the file contains XML configuration which should be loaded on a running instance of ncs with 'ncs_load'.

'json': Specifies that the file contains JSON configuration which should be loaded on a running instance of ncs with 'ncs_load'.

'c-cli': Specifies that the file contains C-style CLI configuration which should be loaded on a running instance of ncs with 'ncs_load'.

'j-cli': Specifies that the file contains J-style CLI configuration which should be loaded on a running instance of ncs with 'ncs_load'.

'i-cli': Specifies that the file contains I-style CLI configuration which should be loaded on a running instance of ncs with 'ncs_load'.

Name

nct-move-device — move devices between cluster device nodes

Synopsis

nct-move-device [*OPTIONS*]

DESCRIPTION

The nct-move-device(1) is a command for moving a managed device from one NCS device node to another using the REST api of NCS. Note that this command require the cluster topology to be described in the 'hostsfile', see the [HOSTSFILE section](#).

OPTIONS

--device <i>DEVICE</i>	The name of the device to move.
--from <i>FROM</i>	The NCS device node name, as specified in the hostsfile.
--to <i>TO</i>	The NCS device node name, as specified in the hostsfile.
--rest-pass <i>RESTPASS</i>	The REST password to be used.
--rest-port <i>RESTPORT</i>	The REST Port number to be used (default: 8888).
--rest-user <i>RESTUSER</i>	The REST user to be used (default: \$USER).
--rest-ssl <i>BOOLEAN</i>	REST via SSL true/false (default: true).

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .

--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
"192.168.23.98", OPTIONS}.
"192.168.23.99", OPTIONS}.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{ "192.168.23.99", [ { name, "pariss"}, ... ] }.  
{ "192.168.23.98", [ { name, "londons"}, ... ] }.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-packages — install NCS packages on host(s)

Synopsis

nct-packages [*OPTIONS*]

DESCRIPTION

With the nct-packages(1) command, it is possible to install, deinstall, load NCS packages and retrieve related package status information, such as if the package is installed, loaded or just installable.

OPTIONS

-c, --cmd <i>CMD</i>	The default <i>CMD</i> is 'list' which will list all the packages in host(s) and their status. The status can be any of 'installable', 'installed' or 'loaded'. These values can also be used as a <i>CMD</i> and will then only show those packages with this status. To get the NCS package moved to host(s), i.e to make it 'installable', use 'fetch' as the <i>CMD</i> . To install or deinstall use the 'install' or 'deinstall' value as the <i>CMD</i> . To load or unload a package into a running NCS use the 'reload' <i>CMD</i> . The 'fetch' command require the '--file' switch to be used. The 'install' and 'deinstall' require the '--package' switch.
--file <i>FILE</i>	The package file name. Example: 'ncs-3.3-tailf-hcc-3.0.8.tar.gz'. This switch is used in combination with the '-c fetch' switch.
--package <i>PACKAGE</i>	The package name. Example: 'ncs-3.3-tailf-hcc-3.0.8'. This switch is used in combination with either '-c install' or '-c deinstall' switches.
--rest-pass <i>RESTPASS</i>	The REST password to be used.
--rest-port <i>RESTPORT</i>	The REST Port number to be used (default: 8888).
--rest-user <i>RESTUSER</i>	The REST user to be used (default: \$USER).
--rest-ssl <i>BOOLEAN</i>	REST via SSL true/false (default: true).

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP[,GROUP]</i>	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the --group switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.

-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups,["service","london"]}, ... ] }.
{ "192.168.24.98", [ {groups,["service","paris"]}, ... ] }.
{ "192.168.23.11", [ {groups,["device","london"]}, ... ] }.
{ "192.168.24.12", [ {groups,["device","paris"]}, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{"192.168.23.99",[{name, "pariss"}, ... ]}.
{"192.168.23.98",[{name, "londons"}, ... ]}.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...
nct stop --name londons --hostsfile ...
```



Name

nct-patch — NCS beam patch handling

Synopsis

nct-patch [*OPTIONS*]

DESCRIPTION

The nct-patch(1) makes it possible to install/remove NCS beam patches as well as display what patches are installed.

OPTIONS

-c, --cmd <i>CMD</i>	The <i>CMD</i> parameter can be any of: 'install', 'remove' or 'show'. To install a beam patch use 'install' together with the '--file' switch. To remove a beam patch use 'remove' with the '--file' switch. To list all installed beam patches, use: 'show' as the <i>CMD</i> .
--file <i>FILE</i>	File name of the NCS beam patch. Example: '/tmp/rest.beam' .
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir</i> /ncs-VSN, allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir</i> /current pointing to one of the ncs-VSN directories. If the --install-dir option is omitted, /opt/ncs will be used for <i>InstallDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the --group switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .

--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example <code>'id_rsa'</code> for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with `[]` brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g `--ssh-user` vs `ssh_user`). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the ‘--groupoper and’ switch which means that the intersection of the specified groups should yield the affected ‘Hosts’. The default of the group mechanism is to use the union if several groups are specified (‘--groupoper or’).

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each *Host*, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the *hostsfile* by a given name if you have added name entries in the *hostsfile*. Study the example below:

```
{ "192.168.23.99", [{name, "pariss"}, ... ] }.  
{ "192.168.23.98", [{name, "londons"}, ... ] }.
```

With the above in your *hostsfile*, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-ssh-cmd — execute a remote ssh command on host(s)

Synopsis

nct-ssh-cmd [*OPTIONS*]

DESCRIPTION

The nct-ssh-cmd(1) is a command to execute a given shell command on one or several host(s)

OPTIONS

-c, --cmd *CMD*

The *CMD* argument is a string that can contain any shell command, or sequence of shell commands, that you want to execute on the host(s). Note that the command(s) are executed as the *SSHUSER* so you may have to run the commands with *sudo* for root preveleges and such. Example: "sudo mv /tmp/foo /etc/bar". Tips: if it is a long running command you may want to use the *nohup* command on the host(s). See the *nohup* man-page.

COMMON OPTIONS

--concurrent *BOOL*

Each *HOST* will be operated upon concurrently. The default value is *true*. If set to *false*, each *HOST* will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one *HOST* at a time.

--name *NAME*

Restrict the command to only operate on the given *NAME*. See the [NAME section](#).

--group *GROUP*[,*GROUP*]

Restrict the command to only operate on the given *GROUP* or groups. See the [GROUPS section](#).

--groupoper *GROUPOPER*

If several groups are specified with the *--group* switch. This switch defines if the union or intersection of those groups should be used. Possible values are: *or* (default) or: *and*. See the [GROUPS section](#).

--help

Will print out a short help text and then terminate the command.

-h, --host *HOST*[,*HOST*]+

Define one *HOST*, or several comma separated *HOST*. The given hosts are the ones the command will operate upon.

--hostsfile *HOSTSFILE*

The *HOSTSFILE* containing the *HOST* entries. See the [HOSTSFILE section](#). By setting the environment variable 'NCT_HOSTSFILE' to *HOSTSFILE* this switch can be omitted.

--progress *BOOL*

If *BOOL* is *true*, then some progress indication will be displayed. The default is *false*.

--ssh-key-name *SSHKEYNAME*

Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the [SSH section](#).

--ssh-pass *SSHPASS*

The SSH password to be used. See also the [SSH section](#).

--ssh-port *SSHPORT*

The SSH Port number to be used (default: 22).

--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.
{ "192.168.23.99", OPTIONS }.
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the `--ssh-key-name` switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{ "192.168.23.99", [ { name, "pariss"}, ... ] }.  
{ "192.168.23.98", [ { name, "londons"}, ... ] }.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-start — start NCS on host(s)

Synopsis

nct-start [*OPTIONS*]

DESCRIPTION

The nct-start(1) will start NCS on host(s) by invoking the ‘/etc/init.d/ncs start’ command. This command is built on top of the ‘nct-check(1)’ command.

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable ‘NCT_HOSTSFILE’ to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example ‘id_rsa’ for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.

-v, --verbose *VERBOSE*

To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.  
{ "192.168.23.99", OPTIONS }.  
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.  
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.  
...
```

The *OPTIONS* is enclosed with *[]* brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,["service","london"]}, ... ]}.  
{ "192.168.24.98", [{groups,["service","paris"]}, ... ]}.  
{ "192.168.23.11", [{groups,["device","london"]}, ... ]}.  
{ "192.168.24.12", [{groups,["device","paris"]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...  
nct stop --group service ...  
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.

**Note**

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.

**Note**

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{"192.168.23.99",[{name, "pariss"}, ... ]}.  
{"192.168.23.98",[{name, "londons"}, ... ]}.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-stop — stop NCS on host(s)

Synopsis

nct-stop [*OPTIONS*]

DESCRIPTION

The nct-stop(1) will stop NCS on host(s) by invoking the ‘/etc/init.d/ncs stop’ command. This command is built on top of the ‘nct-check(1)’ command.

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable ‘NCT_HOSTSFILE’ to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example ‘id_rsa’ for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.

-v, --verbose *VERBOSE*

To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS }.  
{ "192.168.23.99", OPTIONS }.  
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.  
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.  
...
```

The *OPTIONS* is enclosed with `[]` brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the '-' in a switch corresponds to a: '_' in the hostsfile (e.g --ssh-user vs ssh_user). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups,["service","london"]}, ... ] }.  
{ "192.168.24.98", [ {groups,["service","paris"]}, ... ] }.  
{ "192.168.23.11", [ {groups,["device","london"]}, ... ] }.  
{ "192.168.24.12", [ {groups,["device","paris"]}, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...  
nct stop --group service ...  
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the '--groupoper and' switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified ('--groupoper or').

SSH

It is possible to specify the 'SSH User' and 'SSH Password' to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the 'SSH Password' to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use 'SSH KEYS' as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the 'ssh-copy-id' command.

**Note**

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the *--ssh-key-name* switch with any NCT command.

**Note**

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{"192.168.23.99",[{name, "pariss"}, ... ]}.  
{"192.168.23.98",[{name, "londons"}, ... ]}.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```



Name

nct-upgrade — install an NCS release on host(s)

Synopsis

nct-upgrade [*OPTIONS*]

DESCRIPTION

The `nct-upgrade(1)` command will upgrade an NCS release on the specified host(s) using the ‘`--cmd`’ and ‘`--ncs-vsn`’ switches. The default is to first do an NCS backup and to reload any packages. This behaviour can be changed with the ‘`--backup`’ and ‘`--reload_packages`’ switches.

When doing an upgrade, this command will do the following:

- Assert the given NCS version really is installed on the *Host*.
- Assert that given NCS version isn’t already running.
- Make sure that there exist ‘installable’ packages that corresponds to the (old) already installed packages.
- Backup NCS (default action, but optional)
- Deinstall the old packages, then install the new packages.
- Stop NCS
- Rearrange the symlinks under *InstallDir*
- Start NCS (default is to also reload all packages, but optional)

OPTIONS

--backup <i>BOOL</i>	If ‘true’ (default), an NCS backup will be performed before the NCS upgrade takes place. This can be turned off by specifying ‘false’ as the <i>BOOL</i> value.
-c, --cmd <i>CMD</i>	The <i>CMD</i> can be either <i>upgrade</i> (default) or <i>available</i> . The former will perform an NCS upgrade and the latter will show all installed releases on host(s).
--ncs-vsn <i>NCVSN</i>	The <i>NCVSN</i> is a string containing the NCS version number. Example: ‘3.2.1’
--reload-packages <i>BOOL</i>	If ‘true’ (default), any packages will be reloaded when the new (upgraded) NCS is started. This can be turned off by setting the <i>BOOL</i> value to ‘false’.
--rest-pass <i>RESTPASS</i>	The REST password to be used.
--rest-port <i>RESTPORT</i>	The REST Port number to be used (default: 8888).
--rest-user <i>RESTUSER</i>	The REST user to be used (default: \$USER).
--rest-ssl <i>BOOLEAN</i>	REST via SSL true/false (default: true).
--install-dir <i>InstallDir</i>	This is the directory where static files, primarily the code and libraries for the NCS daemon, are installed. The actual directory used for a given invocation of the installation script is <i>InstallDir/ncs-VSN</i> , allowing for coexistence of multiple installed versions. The currently active version is identified by a symbolic link <i>InstallDir/current</i> pointing to one of the <i>ncs-VSN</i> directories. If the <code>--install-dir</code> option is omitted, <i>opt/ncs</i> will be used for <i>InstallDir</i> .

COMMON OPTIONS

--concurrent <i>BOOL</i>	Each <i>HOST</i> will be operated upon concurrently. The default value is <i>true</i> . If set to <i>false</i> , each <i>HOST</i> will be dealt with in sequence. Obviously, the default behaviour often results in a faster total execution time. However, when debugging a command it can be useful to only execute one <i>HOST</i> at a time.
--name <i>NAME</i>	Restrict the command to only operate on the given <i>NAME</i> . See the NAME section .
--group <i>GROUP</i> [, <i>GROUP</i>]	Restrict the command to only operate on the given <i>GROUP</i> or groups. See the GROUPS section .
--groupoper <i>GROUPOPER</i>	If several groups are specified with the <i>--group</i> switch. This switch defines if the union or intersection of those groups should be used. Possible values are: <i>or</i> (default) or: <i>and</i> . See the GROUPS section .
--help	Will print out a short help text and then terminate the command.
-h, --host <i>HOST</i> [, <i>HOST</i>]+	Define one <i>HOST</i> , or several comma separated <i>HOST</i> . The given hosts are the ones the command will operate upon.
--hostsfile <i>HOSTSFILE</i>	The <i>HOSTSFILE</i> containing the <i>HOST</i> entries. See the HOSTSFILE section . By setting the environment variable 'NCT_HOSTSFILE' to <i>HOSTSFILE</i> this switch can be omitted.
--progress <i>BOOL</i>	If <i>BOOL</i> is <i>true</i> , then some progress indication will be displayed. The default is <i>false</i> .
--ssh-key-name <i>SSHKEYNAME</i>	Per default, the default filename of the SSH key pair will be used; for example 'id_rsa' for RSA keys. To override this behaviour you can specify another key name using this switch. See also the SSH section .
--ssh-pass <i>SSHPASS</i>	The SSH password to be used. See also the SSH section .
--ssh-port <i>SSHPORT</i>	The SSH Port number to be used (default: 22).
--ssh-timeout <i>SSHTIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. The SSH timeout is specified in milliseconds. Depending on the type of operation. This timeout may have to be increased.
--ssh-user <i>SSHUSER</i>	The SSH user to be used.
--timeout <i>TIMEOUT</i>	This switch is only needed in case the default timeout, which is <i>infinity</i> , need to be changed. A general timeout which may expire if the ongoing operations takes a too long time to finish. In this case it may be good to increase this value.
-v, --verbose <i>VERBOSE</i>	To increase the output two verbose levels exist. Level 2 will output as much information as possible. Level 1 will output a little less information. This is mostly useful debugging a command.

HOSTSFILE

The *HOSTSFILE* contains entries, called Tuples, looking like:

```
{ "192.168.23.98", OPTIONS } .  
{ "192.168.23.99", OPTIONS } .  
...
```

Note the dot at the end of each Tuple, it is significant. The *OPTIONS* is a list containing options. For example:

```
{ "192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
{"192.168.23.98", [ {ssh_user,"bill"} , ... ] }.
...
```

The *OPTIONS* is enclosed with `[]` brackets. Each option consist of a Tuple with a *KEY* and a *VALUE*. The *KEY* mostly corresponds to a command switch, where the ‘-’ in a switch corresponds to a ‘_’ in the hostsfile (e.g `--ssh-user` vs `ssh_user`). An unary switch is represented as just the switch name.

GROUPS

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [{groups,[ "service", "london" ]}, ... ]}.
{"192.168.24.98", [{groups,[ "service", "paris" ]}, ... ]}.
{"192.168.23.11", [{groups,[ "device", "london" ]}, ... ]}.
{"192.168.24.12", [{groups,[ "device", "paris" ]}, ... ]}.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the ‘--group’ switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the ‘--groupoper and’ switch which means that the intersection of the specified groups should yield the affected 'Hosts'. The default of the group mechanism is to use the union if several groups are specified (‘--groupoper or’).

SSH

It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command.



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the `--ssh-key-name` switch with any NCT command.



Note

For security reasons, it is not recommended to login as *root* on the target machines. Instead, create a user on the target where you install the SSH key, and then use *sudo* to gain root privileges on the target machine.

NAME

You can select a specific host from the hostsfile by a given name if you have added name entries in the hostsfile. Study the example below:

```
{"192.168.23.99",[{name, "pariss"}, ... ]}.  
{"192.168.23.98",[{name, "londons"}, ... ]}.
```

With the above in your hostsfile, you can select a host by name:

```
nct upgrade --name pariss --hostsfile ...  
nct stop --name londons --hostsfile ...
```


Name

nct — a collection of tools that can be used to install and manage NCS nodes.

Introduction

A host running NCS is called an *NCS node*. A host can be either a physical machine or a virtual machine as long as they can be accessed via SSH.

With nct it is possible to install and manage NCS nodes. This assumes that the hosts are running Linux and are accessible via SSH.

The hostsfile

Each NCS node can be operated on independently but the main idea is that nct can operate on a set of NCS nodes. To operate on a set of NCS nodes a, so called, *hostsfile* is needed. A *hostsfile* consists of Host entries according to the following example:

```
{ "192.168.23.99", [ ] }.  
{ "192.168.23.98", [ ] }.  
...etc...
```

Each entry, called a *Tuple*, begin with a '{' bracket and ends with a '}'.



Note

Note the ending dot after the '}' bracket!

Each entry consists of a *Hostname/IP-Address*, enclosed in double quotes and a list of options, where the list begin with a '[' bracket and ends with a corresponding ']' bracket.

In the list of options, information can be given that, in most cases, have a corresponding *switch* option to the various tool commands. So instead of having to specify the SSH User to every tool command with the '--ssh-user' switch; it can be specified in the *hostsfile* as:

```
{ "192.168.23.99", [ {ssh_user, "user"} ] }.  
{ "192.168.23.98", [ {ssh_user, "user"} ] }.  
...etc...
```

Each such switch is represented as a *Tuple* with the switch name and its value, and if it is an unary switch it is represented as just the switch name.



Note

The: '-' in a switch corresponds to a: '_' in the hostsfile. So, for example, the switch '--ssh-user' correspond to 'ssh_user' in the *hostsfile*.

Often it can be useful to be able to group a subset of the *Hosts* in the hostsfile when you want to restrict an operation to only those *Hosts*. This can be done with the group mechanism. Study the example below:

```
{ "192.168.23.99", [ {groups, [ "service", "london" ] }, ... ] }.  
{ "192.168.24.98", [ {groups, [ "service", "paris" ] }, ... ] }.  
{ "192.168.23.11", [ {groups, [ "device", "london" ] }, ... ] }.  
{ "192.168.24.12", [ {groups, [ "device", "paris" ] }, ... ] }.
```

In the example above, we have 4 NCS nodes grouped into two groups named: "london" and: "paris" but also two other groups named: "service" and: "device". Imagine that we may want to do certain operations only on the members in the "london" group or perhaps only on the members in the "device" group. This can easily be achieved by using the '--group' switch to a NCS tools command. For example:

```
nct upgrade --group paris ...
nct stop --group service ...
nct check --group london,device --groupoper and ...
```

In the last example we specify two groups and require the (to be) affected *Hosts* to be member in both groups. This is controlled by the ‘--groupoper and’ switch which means that the intersection of the specified groups should yield the affected ‘Hosts’. The default of the group mechanism is to use the union if several groups are specified (‘--groupoper or’).

For a complete list of options available for a hostsfile, see `nct-hostsfile(1)`

The use of SSH

The NCS tools make heavy use of SSH for running commands and copying file on/to the *Hosts*. It is possible to specify the ‘SSH User’ and ‘SSH Password’ to be used for each Host, either with a switch to a command or in the *hostsfile*. It is recommended to add the ‘SSH Password’ to the *hostsfile* and prohibit other users read access to the file for security reasons.

It is also possible to use ‘SSH KEYS’ as long as they do not require a passphrase.

Then, for each *Host*, setup the SSH key authentication. This can easily be done with the ‘ssh-copy-id’ command. Example:

```
ssh-copy-id user@192.168.23.99
```



Note

Per default, the default filename of the SSH key pair will be used; for example *id_rsa* for RSA keys. To override this behaviour you can use the ‘--ssh-key-name <keyname>’ switch with any NCT command.

Verify SSH access

Assume we have a hostsfile looking like this:

```
{ "192.168.23.99", [ {groups, [ "service", "paris" ] }, {ssh_user, "user" } ] }.
{ "192.168.23.98", [ {groups, [ "service", "london" ] }, {ssh_user, "user" } ] }.
{ "192.168.23.11", [ {groups, [ "device", "paris" ] }, {ssh_user, "user" } ] }.
{ "192.168.23.12", [ {groups, [ "device", "paris" ] }, {ssh_user, "user" } ] }.
{ "192.168.23.21", [ {groups, [ "device", "london" ] }, {ssh_user, "user" } ] }.
{ "192.168.23.22", [ {groups, [ "device", "london" ] }, {ssh_user, "user" } ] }.
```

where we have setup SSH key authentication on all the hosts as explained earlier. The *Hosts* are running Linux and the prerequisites has been installed.

We can now verify that the nct can access all the *Hosts* using the following command:

```
nct check --hostsfile hostsfile -c ssh
```

If successful, the command may return something like:

```
....
SSH Check to 192.168.23.99:22
SSH OK : 'ssh uname' returned: Linux
....
```

We also need to verify that we can become root on the Hosts by executing the ‘sudo’ command. This can be checked like this:

```
nct check --hostsfile hostsfile -c ssh-sudo
```

If successful, the command may return something like:

```
....
SSH+SUDO Check to 192.168.23.99:22
SSH+SUDO OK
....
```

If you run the 'nct check' without the '-c' switch it will actually try to verify the following:

- SSH access
- SUDO access
- DISK USAGE with a configurable warning limit
- REST interface is up
- NETCONF interface is up
- What NCS version is running
- Is HA enabled

It can look something like this:

```
....
ALL Check to 192.168.22.99:22
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE <WARNING> FileSys=/dev/sda4 (/var,/opt) Use=89%
REST OK
NETCONF OK
NCS-VSN : 3.3
HA : mode=master, node-id=master_primary, connected-slave=master_secondary
....
```

To run any Linux command on the *Hosts*, use the *nct ssh-cmd*.

Example:

```
nct ssh-cmd --hostsfile hostsfile \  
-c "sudo sh -c 'yes | /opt/ncs/current/bin/ncs-uninstall --all'"
```

Install NCS

Assume we have verified that SSH access works as shown above.

We can then install NCS on all the *Hosts* like this:

```
nct install --hostsfile hostsfile --file ncs-3.3.linux.x86_64.installer.bin
```

The command will return before the installation is completed. We can now check the progress by displaying the content of the install log on each of the *Hosts* like this:

```
nct install --hostsfile hostsfile -c check --file ncs-3.3.linux.x86_64.installer.bin
```

The output, when the installation is completed will look something like this:

```
....
Check Install of NCS to 192.168.23.99:22
>> Content of : /tmp/ncs-3.3.linux.x86_64.installer.bin.log
INFO Using temporary directory /tmp/ncs_installer.17571 to stage NCS installation bundle
INFO Using /opt/ncs/ncs-3.3 for static files
INFO Unpacked ncs-3.3 in /opt/ncs/ncs-3.3
INFO Found and unpacked corresponding DOCUMENTATION_PACKAGE
INFO Found and unpacked corresponding EXAMPLE_PACKAGE
```

```

INFO  Generating default SSH hostkey (this may take some time)
INFO  SSH hostkey generated
INFO  Environment set-up generated in /opt/ncs/ncs-3.3/ncsrc
INFO  NCS installation script finished
INFO  Found and unpacked corresponding NETSIM_PACKAGE
INFO  NCS installation complete
....

```

Note that the first time(s) you run the ‘-c check’ command, the installation may not yet be finished and you will only see a few of the lines above. Then wait a short while before re-run the command. When all log output show the line ‘NCS installation complete’ you’re done.

If you want to limit the check to just one (or a few hosts) you can specify them on the command line like this:

```

nct install -h 192.168.23.99,192.168.23.98 --ssh-user user \
c check --file ncs-3.3.linux.x86_64.installer.bin

```

If the installation fail, on one or several hosts, then try the following:

- Do the log output give you any hint on what went wrong?
- Increase the verbosity of the command output with the -v 2 switch.
- Is it possible to install ncs manually on the *Host* (try: `sh -c "ncs-3.3.linux.x86_64.installer.bin --system-install"`).

If nothing works, get in contact with the nct team for more help.

If the installations was successful you can start NCS like this:

```

nct start --hostsfile hostsfile

```

Finally, check that NCS actually is up and running:

```

nct check --hostsfile hostfile

```

Adding NEDs and packages

When NCS is installed and is up and running, it is time to add NEDs/packages to the systems. The name of a package is defined as:

```

ncs-<NCSVSN>-<PACKAGE><VSN>.tar.gz

```

Where *NCSVSN* is the major version of NCS used to build the package, *PACKAGE* is the name of the package and *VSN* the version of the package.

First we need to put a package on each *Host*:

```

nct packages --hostsfile hostsfile -c fetch \
--file ncs-3.3-tailf-hcc-3.0.8.tar.gz

```

If successful, the output could look like this:

```

....
Fetch Package at 192.168.23.99:8080
OK
....

```

To check the package status we can run:

```

nct packages --hostsfile hostsfile

```

Which may result in the following output:

```
....
Package Info at 192.168.22.99:8080
ncs-3.3-tailf-hcc-3.0.8 (installable)
....
```

The status information above, shown between the parentheses, can take on one of the following values:

installable	The package is ready to be installed
installed	The package has been installed
loaded	The package has been loaded into the NCS node

To install the package we can run:

```
nct packages --hostsfile hostsfile -c install \
--package ncs-3.3-tailf-hcc-3.0.8
```

If we again check the the status, we will see the following:

```
....
Package Info at 192.168.22.99:8080
ncs-3.3-tailf-hcc-3.0.8 (installed)
....
```

What remains is to actually load the package into the NCS node, which we can do like this:

```
nct packages --hostsfile hostsfile -c reload
```

And if the reload is successful we may get in return:

```
....
Reload Packages at 192.168.22.99:8080
tailf-hcc          true
....
```

In case the the reload would fail we would get something like:

```
....
Reload Packages at 192.168.22.99:8080
tailf-hcc          false ...some text here...
....
```

If we yet again check the the status, we will see the following:

```
....
Package Info at 192.168.22.99:8080
tailf-hcc-3.0.8 (loaded)
ncs-3.3-tailf-hcc-3.0.8 (installed)
....
```

It is of course possible to deinstall a package:

```
nct packages --hostsfile hostsfile -c deinstall \
--package ncs-3.3-tailf-hcc-3.0.8
```

A status check will reveal the following:

```
....
Package Info at 192.168.22.99:8080
tailf-hcc-3.0.8 (loaded)
ncs-3.3-tailf-hcc-3.0.8 (installable)
```

....

As you can see, the package is still loaded, to unload it we need to do a *reload* again:

```
nct packages --hostsfile hostsfile -c reload
```

A final status check will now show us this:

```
....
Package Info at 192.168.22.99:8080
ncs-3.3-tailf-hcc-3.0.8 (installable)
....
```

Upgrade a NED/package

When to upgrade a package to a new version it is recommended to first make an NCS backup:

```
nct backup --hostsfile hostsfile
```

The result output may look something like:

```
....
SSH Check to 192.168.23.22:22
SSH OK : 'ssh sudo /opt/ncs/current/bin/ncs-backup --non-interactive' \
returned: INFO Backup /var/opt/ncs/backups/ncs-3.3@2014-12-05T19:47:58.backup\
created successfully
....
```

As the header of the output above reveal, the ‘nct backup’ command is really just using the versatile ‘nct check’ command to execute the NCS backup command. The important part to note however is the path to the successfully created backup file.

After this successful NCS backup we can upgrade a NED package as:

```
nct packages --hostsfile hostsfile -c deinstall \
--package ncs-3.3-tailf-hcc-3.0.8

nct packages --hostsfile hostsfile -c fetch \
--file ncs-3.3-tailf-hcc-3.0.9.tar.gz

nct packages --hostsfile hostsfile -c install \
--package ncs-3.3-tailf-hcc-3.0.9

nct packages --hostsfile hostsfile -c reload
```

Upgrading NCS

When upgrading NCS to another version we make use of the ‘nct upgrade’ command. This command will perform the following actions:

- Assert the given NCS version really is installed on the *Host*.
- Assert that given NCS version isn’t already running.
- Make sure that there exist ‘installable’ packages that corresponds to the (old) already installed packages.
- Backup NCS (default action, but optional)
- Deinstall the old packages, then install the new packages.
- Stop NCS
- Rearrange the symlinks under */opt/ncs*
- Start NCS (default is to also reload all packages, but optional)

Invoking it can look something like this:

```
nct upgrade --hostsfile hostsfile --ncs-vsn 3.2.1.1
```

and the corresponding output can look like this:

```
....
Upgrade NCS to 192.168.23.99
OK : upgrade done
....
```

In case we do a major version upgrade, let's say from 3.3 to 3.4, the command will make sure that all 3.3 packages are uninstalled before the upgrade can take place. Example:

```
$ nct upgrade --hostsfile hostsfile --ncs-vsn 3.4
```

which may result in an error like this:

```
....
ERROR: packages that need to be uninstalled on 192.168.23.99:
ncs-3.3-modok-1.0
....
```

The remedy to this is to first run the 'nct packages' command to 'deinstall' the old packages, before doing the upgrade.



Note

It is very important that any corresponding new packages are installed before the upgrade is performed. If not, you will lose all the package specific configuration at reload of the NCS packages.

Installing beam patches

It is possible to patch NCS with, so called, '.beam' files that are updated versions of NCS internal (binary) program code.

Let's say we want to install a patch file named: 'rest.beam'. It can look like this:

```
nct patch --hostsfile hostsfile --file rest.beam
```

and the corresponding output:

```
....
Install NCS patch to 192.168.23.99
>> OK
....
```

To see which patches that are installed you can use the '-c show' switch like this:

```
nct patch --hostsfile hostsfile -c show
```

and the output looks something like this:

```
....
Show loaded NCS patches to 192.168.23.99
NAME                VERSION                COMPILE TIME
-----
rest                41da353169d431bbbf5417b85e5d0d06  26-Nov-2014::12:56:11
....
```

To remove a patch (and reload the original code), use the '-c remove' switch like this:

```
nct patch --hostsfile hostsfile --file rest.beam -c remove
```

Running CLI commands

It is possible to run any NCS CLI command over a set of NCS nodes. For example, to list all cleared alarms on a group of nodes:

```
nct cli-cmd --hostsfile hostsfile --group paris \  
-c "show alarms alarm-list alarm ncs * * * is-cleared"
```

Operate on HA nodes



Note

This command requires the HA Controller: 'tailf-hcc', and a working HA configuration. You can read more about NCS High Availability feature in the User Guide for NCS, and in the deployment document for tailf-hcc.

This command allows you to preform actions available in the tailf-hcc package.

For example, you can activate your HA environment with the command:

```
nct ha --action cluster-activate --hostsfile hostsfile --group service
```

Move a device between NCS nodes

The 'nct move-device' is a command for moving a device managed by one NCS node, to another NCS node, both in a cluster setup and a non-cluster setup.

This command requires a hostsfile, which must contain the (cluster) topology. Each host in the hostsfile requires the option {name,<name>}, each device node requires the option {service_node,<name>} and each service node requires the option {device_nodes,[{<name>,<remote-name>} ...]}, where <name> is a unique identifier for the node in the hostsfile and <remote-name> the configured name for the cluster remote-node on the service nodes.

A valid configuration for a service node and a device node would look something like this:

```
{ "192.168.23.99",  
  [{name,"pariss"}],  
  {device_nodes,[{"parisd1","d1"}, {"parisd2","d2"}]}}.
```

```
{ "192.168.23.11",  
  [{name,"parisd1"}],  
  {service_node,"pariss"}}.
```

The command is executed like this:

```
nct move-device --from parisd1 -to parisd2 \  
--device m0 --hostfile hostfile
```

Node *parisd1* and *parisd2* are two NCS device nodes.

Man pages

Each nct command has a corresponding man page.

Example:

```
man nct-install
```


SEE ALSO

nct-backup(1)

nct-check(1)

nct-cli-cmd(1)

nct-copy(1)

nct-get-logs(1)

nct-ha(1)

nct-install(1)

nct-load-config(1)

nct-move-device(1)

nct-packages(1)

nct-patch(1)

nct-ssh-cmd(1)

nct-start(1)

nct-stop(1)

nct-upgrade(1)



Name

pyang — validate and convert YANG modules to various formats

Synopsis

```
pyang [--verbose] [--canonical] [--strict] [--ietf] [--lax-xpath-checks] [--hello] [--check-update-from
oldfile] [-o outfile] [-f format] [-p path] [-W warning] [-E error] [-a filename] [--no-
features] | [-F feature]] [--ignore-unknown-features] file ...
```

```
pyang --tailf-sanitize [--tailf-remove-body] [--tailf-keep-actions] [--tailf-keep-info] [--tailf-keep-tailf-
typedefs] [--tailf-keep-symlink-when] [--tailf-keep-symlink-must] [--tailf-keep-display-when]
```

```
pyang -h | --help
```

```
pyang --rest-doc [--rest-doc-db] [--rest-doc-address] [--rest-doc-output] [--rest-doc-query-type] [--rest-
doc-end-list-url-with-keys] [--rest-doc-end-list-url-with-no-keys] [--rest-doc-list-keys] [--rest-doc-path] [--
rest-doc-curl-flags]
```

```
pyang -v | --version
```

One or more *file* parameters may be given on the command line. They denote either YANG modules to be processed (in YANG or YIN syntax) or, using the `--hello` switch, a server <hello> message conforming to [RFC 6241](#) and [RFC 6020](#), which completely defines the data model - supported YANG modules as well as features and capabilities. In the latter case, only one *file* parameter may be present.

If no files are given, **pyang** reads input from stdin, which must be one module or a server <hello> message.

Description

The **pyang** program is used to validate YANG modules ([RFC 6020](#)). It is also used to convert YANG modules into equivalent YIN modules. From a valid module a hybrid DSDL schema ([RFC 6110](#)) can be generated.

If no *format* is given, the specified modules are validated, and the program exits with exit code 0 if all modules are valid.

Options

<code>-h --help</code>	Print a short help text and exit.
<code>-v --version</code>	Print the version number and exit.
<code>-e --list-errors</code>	Print a listing of all error codes and messages pyang might generate, and then exit.
<code>--print-error-code</code>	On errors, print the symbolic error code instead of the error message.
<code>-Werror</code>	Treat warnings as errors.
<code>-Wnone</code>	Do not print any warnings.
<code>-Werrorcode</code>	Treat <i>errorcode</i> as a warning, even if <code>-Werror</code> is given. <i>errorcode</i> must be a warning or a minor error.
	Use <code>--list-errors</code> to get a listing of all errors and warnings.
	The following example treats all warnings except the warning for unused imports as errors:

<code>-E <i>errorcode</i></code>	<pre>\$ pyang --Werror -W UNUSED_IMPORT</pre> <p>Treat the warning <i>errorcode</i> as an error.</p> <p>Use <code>--list-errors</code> to get a listing of all errors and warnings.</p> <p>The following example treats only the warning for unused import as an error:</p> <pre>\$ pyang --Werror -W UNUSED_IMPORT</pre>
<code>--ignore-errors</code>	Ignore errors. Use with care. Plugins that don't expect to be invoked if there are errors present may crash.
<code>--keep-comments</code>	This parameter has effect only if a plugin can handle comments. One example of such a plugin is the 'yang' output format plugin.
<code>--canonical</code>	Validate the module(s) according to the canonical YANG order.
<code>--strict</code>	Force strict YANG compliance. Currently this checks that the <code>deref()</code> function is not used in XPath expressions and leafrefs.
<code>--ietf</code>	Validate the module(s) according to IETF rules as specified in RFC 6087 . In addition, it checks that the module is in canonical order, and that <code>--max-line-length</code> is 72 so that the module fits into an RFC.
<code>--lax-xpath-checks</code>	Lax checks of XPath expressions. Specifically, do not generate an error if an XPath expression uses a variable or an unknown function.
<code>-L --hello</code>	Interpret the input file or standard input as a server <hello> message. In this case, no more than one <i>file</i> parameter may be given.
<code>--trim-yin</code>	In YIN input modules, remove leading and trailing whitespace from every line in the arguments of the following statements: 'contact', 'description', 'error-message', 'organization' and 'reference'. This way, the XML-indented argument texts look tidy after translating the module to the compact YANG syntax.
<code>--max-line-length <i>maxlen</i></code>	Give a warning if any line is longer than <i>maxlen</i> .
<code>--max-identifier-length <i>maxlen</i></code>	Give a error if any identifier is longer than <i>maxlen</i> .
<code>-f --format <i>format</i></code>	Convert the module(s) into <i>format</i> . Some translators require a single module, and some can translate multiple modules at one time. If no <i>outfile</i> file is specified, the result is printed on stdout. The supported formats are listed in Output Formats below.
<code>-o --output <i>outfile</i></code>	Write the output to the file <i>outfile</i> instead of stdout.
<code>--features <i>features</i></code>	<p><i>features</i> is a string on the form <i>modulename</i>: [<i>feature</i>(,<i>feature</i>)*]</p> <p>This option is used to prune the data model by removing all nodes that are defined with a "if-feature" that is not listed as <i>feature</i>. This option affects all output formats.</p> <p>This option can be given multiple times, and may be also combined with <code>--hello</code>. If a <code>--features</code> option specifies different supported features for a module than the hello file, the <code>--features</code> option takes precedence.</p> <p>If this option is not given, nothing is pruned, i.e., it works as if all features were explicitly listed.</p>

`--deviation-module file`

For example, to view the tree output for a module with all if-feature'd nodes removed, do:

```
$ pyang -f tree --features mymod: mymod.yang
```

This option is used to apply the deviations defined in *file*. This option affects all output formats.

This option can be given multiple times.

`-p --path path`

For example, to view the tree output for a module with some deviations applied, do:

```
$ pyang -f tree --deviation-module mymod-devs.yang mymod.yang
```

path is a colon (:) separated list of directories to search for imported modules. This option may be given multiple times.

The following directories are always added to the search path:

- 1 current directory
- 2 \$YANG_MODPATH
- 3 \$HOME/yang/modules
- 4 \$YANG_INSTALL/yang/modules OR if \$YANG_INSTALL is unset <the default installation directory>/yang/modules (on Unix systems: /usr/share/yang/modules)

`--plugindir plugindir`

Load all YANG plugins found in the directory *plugindir*. This option may be given multiple times.

list of directories to search for pyang plugins. The following directories are always added to the search path:

- 1 pyang/plugins from where pyang is installed
- 2 \$PYANG_PLUGINPATH

`--check-update-from
oldfile`

Checks that a new revision of a module follows the update rules given in [RFC 6020](#). *oldfile* is the old module and *file* is the new version of the module.

If the old module imports or includes any modules or submodules, it is important that the old versions of these modules and submodules are found. By default, the directory where *oldfile* is found is used as the only directory in the search path for old modules. Use the option `--check-update-from-path` to control this path.

`-P --check-update-from-path
oldpath
file...`

oldpath is a colon (:) separated list of directories to search for imported modules. This option may be given multiple times.

These are the names of the files containing the modules to be validated, or the module to be converted.

Tail-f Specific Options

`-a --annotate filename`

filename is the name of a file containing a YANG module with tailf:annotate statements.

This parameter can be given multiple times.

`--cs-feature feature`

Indicates that support for *feature* should be present in the generated confspec file. If *feature* is defined in an imported YANG module, it must be given as *prefix:feature-name*.

If no such parameter is given, all features in the module is supported.

This parameter can be given multiple times.

`--cs-no-features`

Indicates that no features are supported.

`--cs-ignore-unknown-features`

Instructs the compiler to not give an error if an unknown feature is specified with `--feature`.

`--cs-use-description`

Normally, 'description' statements are ignored by pyang when translating to confspecs. Instead the 'tailf:info' statement is used as help and information text in the CLI and WebUI. When this option is specified, text in 'description' statements is used if no 'tailf:info' statement is present.

Tail-f Sanitation Options

`--tailf-sanitize`

This option removes all tailf specific statements from a module, including the import statement that imports tailf-common, if possible. This option is intended to be used together with `-f yang` or `-f yin`, in order to produce a module without any references to tailf modules.

This option is useful in order to publish YANG modules to NETCONF clients, like Tail-f's NCS.

The behavior of this option can be changed with the other options listed in this manual page.

By default, this option:

- Removes all tailf extension statements. Most of these extensions are used to control the implementation on the server, or to tweak the CLI experience. These extensions are not needed in a NETCONF client.
- Removes any 'must' or 'when' statement that uses a non-standard XPath function.
- Removes any node marked with a 'tailf:hidden full' statement. Such a node is not visible in any agent.
- Inlines the subtree that a 'tailf:symlink' points to. If any node in the target subtree contains 'when' or 'must' statements, these are removed. The reason for this is that in general, it is not possible to guarantee that the argument to such a statement will point to a node that is visible in the module.
- Copies any typedefs from 'tailf-common' into the module, and changes any references to such types to reference the copied type.

`--tailf-remove-body`

Removes all 'container', 'list', 'leaf', 'leaf-list', 'augment', 'rpc', and 'notification' statements from the module.

This can be useful if the module is normally compiled with the `--export` option to **confdc**, and the module is not exported to NETCONF, but it still has typedefs, groupings, features, or identities

`--tailf-keep-non-std-xpath`

`--tailf-keep-actions`

`--tailf-keep-info`

`--tailf-keep-tailf-typedefs`

`--tailf-keep-symlink-when`

`--tailf-keep-symlink-must`

`--tailf-keep-display-when`

that are referenced by some other module. By using this option, such a module can safely be published to a NETCONF client.

In the sanitation process, do not remove 'must' and 'when' statements with non-standard XPath functions.

If a module is prepared for NCS, this option should be used, so that NCS can make use of the 'must' and 'when' statements.

In the sanitation process, do not remove 'tailf:action' statements. This is useful if the client understands this extension.

If a module is prepared for NCS, this option should be used, so that NCS can invoke actions on the server.

In the sanitation process, do not remove 'tailf:info' statements. This is useful if the client understands this extension.

If a module is prepared for NCS, this option should be used, so that the NCS CLI prints the given help text.

In the sanitation process, do not copy types from 'tailf-common'. This option can be given if the 'tailf-common' module is published to the client.

If a module is prepared for NCS, this option should be used, so that the NCS uses the correct internal representation of these types.

In the sanitation process, do not remove 'when' statements found in the 'tailf:symlink' target subtree. Use this option only if the 'when' expressions do not escape out of the symlinked subtree.

In the sanitation process, do not remove 'must' statements found in the 'tailf:symlink' target subtree. Use this option only if the 'must' expressions do not escape out of the symlinked subtree.

In the sanitation process, do not remove 'tailf:display-when' statements. This is useful if the client understands this extension.

REST Documentation Options

`--rest-doc-db=[candidate | running]`
`--rest-doc-address=address`
`--rest-doc-output=[delete | delete_curl | docbook | get | get_curl | patch | patch_curl | post | post_curl | put | put_curl]`

This option defines what database to use, default is *running*.

Address used in curl output, default is *http://localhost:8080*.

This option defines the output format, default is *get*.

- *delete* output generic DELETE format.
- *delete_curl* output curl DELETE format.
- *docbook* output docbook format.

The docbook format contains delete, delete_curl, get, get_curl, patch, patch_curl, put, put_curl, post and post_curl formats for all the top containers and lists in a module.

- *get* output generic GET format.
- *get_curl* output curl GET format.
- *patch* generates the generic PATCH format.
- *patch_curl* generates the curl PATCH format.
- *post* generates the generic POST format.
- *post_curl* generates the curl POST format.

	<ul style="list-style-type: none"> • <i>put</i> generates the generic PUT format. • <i>put_curl</i> generates the curl PUT format.
<code>--rest-doc-query-type=[default shallow deep]</code>	GET result depth, default is <i>default</i> .
<code>--rest-doc-end-list-url-with-keys</code>	End example list path with key parameters. This is the default.
<code>--rest-doc-end-list-url-with-no-keys</code>	Don't end example list path with key parameters.
<code>--rest-doc-list-keys</code>	JSON dictionary of list keys and values. Replaces the example leaf- and key-values in GET, PATCH, POST and PUT, with the specified values in the dictionary.
<code>--rest-doc-path</code>	Key path to the yang-statement to generate output for.
<code>--rest-doc-curl-flags</code>	Additional curl-flags added to the <code>--rest-doc-output *_curl</code> commands.

OUTPUT EXAMPLES

<code>--rest-doc-output=get</code>	# leaf (string) /t1:test/t1:leaf-j-1/t1:s1 GET /api/running/test/leaf-j-1/s1 Accept: application/vnd.yang.data+json
<code>--rest-doc-output=get_curl</code>	curl -X GET \ -u admin:admin \ -H "Accept: application/vnd.yang.data+json" \ http://localhost:8080/api/running/test/leaf-j-1/s1
<code>--rest-doc-output=patch_curl</code>	echo ' { "cont-j-1" : { "s1" : "s1 value", "i1" : 42, "u1" : 42 } } ' curl -X PATCH -d @- -u admin:admin \ -H "Content-type: application/vnd.yang.data+json" \ http://localhost:8080/api/running/test/cont-j-1

Output Formats

Installed **pyang** plugins may define their own options, or add new formats to the `-f` option. These options and formats are listed in **pyang -h**.

<i>capability</i>	Capability URIs for each module of the data model.
<i>depend</i>	Makefile dependency rule for the module.
<i>dsdl</i>	Hybrid DSDL schema, see RFC 6110 .
<i>hypertree</i>	Hyperbolic tree navigator that can be displayed by treebolic .
<i>jsonxsl</i>	XSLT stylesheet for transforming XML instance documents to JSON.
<i>jstree</i>	HTML/JavaScript tree navigator.
<i>jtox</i>	Driver file for transforming JSON instance documents to XML.
<i>omni</i>	An applescript file that draws a diagram in OmniGraffle .
<i>sample-xml-skeleton</i>	Skeleton of a sample XML instance document.
<i>tree</i>	Tree structure of the module.
<i>uml</i>	UML file that can be read by plantuml to generate UML diagrams.

<i>xmi</i>	XMI file that can be imported by ArgoUML .
<i>xsd</i>	DEPRECATED: W3C XML Schema.
<i>yang</i>	Normal YANG syntax.
<i>yin</i>	The XML syntax of YANG.
<i>cs</i>	Tail-f confspec language

Capability Output

The *capability* prints a capability URL for each module of the input data model, taking into account features and deviations, as described in section 5.6.4 of [RFC 6020](#).

Options for the *capability* output format:

`--capability-entity` Write ampersands in the output as XML entities ("&").

Depend Output

The *depend* output generates a Makefile dependency rule for files based on a YANG module. This is useful if files are generated from the module. For example, suppose a *.c* file is generated from each YANG module. If the YANG module imports other modules, or includes submodules, the *.c* file needs to be regenerated if any of the imported or included modules change. Such a dependency rule can be generated like this:

```
$ pyang -f depend --depend-target mymod.c \
    --depend-extension .yang mymod.yang
mymod.c : ietf-yang-types.yang my-types.yang
```

Options for the *depend* output format:

<code>--depend-target</code>	Makefile rule target. Default is the module name.
<code>--depend-extension</code>	YANG module file name extension. Default is no extension.
<code>--depend-no-submodules</code>	Do not generate dependencies for included submodules.
<code>--depend-include-path</code>	Include file path in the prerequisites. Note that if no <code>--depend-extension</code> has been given, the prerequisite is the filename as found, i.e., ending in ".yang" or ".yin".
<code>--depend-ignore-module</code>	Name of YANG module or submodule to ignore in the prerequisites. This option can be given multiple times.

DSDL Output

The *dsdl* output takes a data model consisting of one or more YANG modules and generates a hybrid DSDL schema as described in [RFC 6110](#). The hybrid schema is primarily intended as an interim product used by **yang2dsdl(1)**.

The *dsdl* plugin also supports metadata annotations, if they are defined and used as described in [draft-lhotka-netmod-yang-metadata](#).

Options for the *dsdl* output format:

<code>--dsdl-no-documentation</code>	Do not print documentation annotations
<code>--dsdl-no-dublin-core</code>	Do not print Dublin Core metadata terms
<code>--dsdl-record-defs</code>	Record translations of all top-level typedefs and groupings in the output schema, even if they are not used. This is useful for translating library modules.

hypertree output

The *hypertree* output generates a hyperbolic YANG browser. The generated xml file can be imported to **treebolic** (<http://treebolic.sourceforge.net/en/>).

Color coding in the tree:

- Light green node background : config = True
- Light yellow node background : config = False
- Red node foreground : mandatory = True
- White leaf node background : index
- Orange foreground : presence container

The xml file references an images folder that needs to exist in the same folder as the generated file. This is installed as share/yang/images in the pyang installation directory. The easiest way is to symlink to this directory.

```
pyang -f hypertree model.yang -o model.xml
```

Prepare a HTML file that links to the generated XMI file:

```
<applet code="treebolic.applet.Treebolic.class"
archive="TreebolicAppletDom.jar"
id="Treebolic" width="100%" height="100%">
<param name="doc" value="model.xml">
</applet>
```

hypertree output specific option:

<code>--hypertree-help</code>	Print help on hypertree usage and exit.
<code>--xmi-no-assoc-names</code>	Do not print association names. ArgoUML has no way of hiding the association name and the diagram gets cluttered.

JSONXSL Output

The *jsonxsl* output generates an XSLT 1.0 stylesheet that can be used for transforming an XML instance document into JSON text as specified in [draft-ietf-netmod-yang-json](#). The XML document must be a valid instance of the data model which is specified as one or more input YANG modules on the command line (or via a <hello> message, see the `--hello` option).

The data tree(s) must be wrapped at least in either <nc:data> or <nc:config> element, where "nc" is the namespace prefix for the standard NETCONF URI "urn:ietf:params:xml:ns:netconf:base:1.0", or the XML instance document has to be a complete NETCONF RPC request/reply or notification. Translation of RPCs and notifications defined by the data model is also supported.

The generated stylesheet accepts the following parameters that modify its behaviour:

- *compact*: setting this parameter to 1 results in a compact representation of the JSON text, i.e. without any whitespace. The default is 0 which means that the JSON output is pretty-printed.
- *ind-step*: indentation step, i.e. the number of spaces to use for each level. The default value is 2 spaces. Note that this setting is only useful for pretty-printed output (compact=0).

The stylesheet also includes the file `jsonxsl-templates.xsl` which is a part of **pyang** distribution.

jstree Output

The *jstree* output generates an HTML/JavaScript page that presents a tree-navigator to the given YANG module(s).

jstree output specific option:

`--jstree-no-path` Do not include paths in the output. This option makes the page less wide.

JTOX Output

The *jtox* output generates a driver file which can be used as one of the inputs to **json2xml** for transforming a JSON document to XML as specified in [draft-ietf-netmod-yang-json](#).

The *jtox* output itself is a JSON document containing a concise representation of the data model which is specified as one or more input YANG modules on the command line (or via a <hello> message, see the `--hello` option).

See **json2xml** manual page for more information.

Omni Output

The plugin generates an applescript file that draws a diagram in OmniGraffle. Requires OmniGraffle 6. Usage:

```
$ pyang -f omni foo.yang -o foo.scpt
$ osascript foo.scpt
```

omni output specific option:

`--omni-path path` Subtree to print. The *path* is a slash ("/") separated path to a subtree to print. For example `"/nacm/groups"`.

Sample-xml-skeleton Output

The *sample-xml-skeleton* output generates an XML instance document with sample elements for all nodes in the data model, according to the following rules:

- An element is present for every leaf, container or anyxml.
- At least one element is present for every leaf-list or list. The number of entries in the sample is min(1, min-elements).
- For a choice node, sample element(s) are present for each case.
- Leaf, leaf-list and anyxml elements are empty (but see the `--sample-xml-skeleton-defaults` option below).

Note that the output document will most likely be invalid and needs manual editing.

Options specific to the *sample-xml-skeleton* output format:

<code>--sample-xml-skeleton-doctype=type</code>	Type of the sample XML document. Supported values for <i>type</i> are <i>data</i> (default) and <i>config</i> . This option determines the document element of the output XML document (<data> or <config> in the NETCONF namespace) and also affects the contents: for <i>config</i> , only data nodes representing configuration are included.
<code>--sample-xml-skeleton-defaults</code>	Add leaf elements with defined defaults to the output with their default value. Without this option, the default elements are omitted.

`--sample-xml-skeleton-
annotations`

Add XML comments to the sample documents with hints about expected contents, for example types of leaf nodes, permitted number of list entries etc.

Tree Output

The *tree* output prints the resulting schema tree from one or more modules. Use **pyang --tree-help** to print a description on the symbols used by this format.

Tree output specific options:

`--tree-help`

Print help on symbols used in the tree output and exit.

`--tree-depth depth`

Levels of the tree to print.

`--tree-path path`

Subtree to print. The *path* is a slash ("/") separated path to a subtree to print. For example `"/nacm/groups"`.

UML Output

The *uml* output prints an output that can be used as input-file to **plantuml** (<http://plantuml.sourceforge.net>) in order to generate a UML diagram. Note that it requires **graphviz** (<http://www.graphviz.org/>).

For large diagrams you may need to increase the Java heap-size by the `-XmxSIZEm` option, to java. For example: **java -Xmx1024m -jar plantuml.jar**

Options for the *UML* output format:

`--uml-classes-only`

Generate UML with classes only, no attributes

`--uml-split-pages=layout`

Generate UML output split into pages, NxN, example 2x2. One .png file per page will be rendered.

`--uml-output-
directory=directory`

Put the generated .png files(s) in the specified output directory. Default is "img"

`--uml-title=title`

Set the title of the generated UML diagram, (default is YANG module name).

`--uml-header=header`

Set the header of the generated UML diagram.

`--uml-footer=footer`

Set the footer of the generated UML diagram.

`--uml-long-identifiers`

Use complete YANG schema identifiers for UML class names.

`--uml-no=arglist`

This option suppresses specified arguments in the generated UML diagram. Valid arguments are: uses, leafref, identity, identityref, typedef, annotation, import, circles, stereotypes. Annotation suppresses YANG constructs rendered as annotations, examples module info, config statements for containers. Example `--uml-no=circles,stereotypes,typedef,import`

`--uml-truncate=elemlist`

Leafref attributes and augment elements can have long paths making the classes too wide. This option will only show the tail of the path. Example `--uml-truncate=augment,leafref`.

`--uml-inline-groupings`

Render the diagram with groupings inlined.

`--uml-inline-augments`

Render the diagram with augments inlined.

`--uml-max-enums=number`

Maximum of enum items rendered.

`--uml-filter-file=file`

NOT IMPLEMENTED: Only paths in the filter file will be included in the diagram. A default filter file is generated by option `--filter`.

XMI Output

The *xmi* output prints an XMI file that can be imported by ArgUML <http://argouml.tigris.org/>.

Drag all classes to the diagram area in ArgoUML and use the Arrange-Layout menu.

XMI output specific option:

XSD Output

NOTE: The XSD output plugin is deprecated. Use the dsdl plugin instead.

Options for the *xsd* output format:

<code>--xsd-no-appinfo</code>	Do not print YANG specific appinfo.
<code>--xsd-no-lecture</code>	Do not print the lecture about how the XSD can be used.
<code>--xsd-no-imports</code>	Do not generate any <code>xs:imports</code> .
<code>--xsd-break-pattern</code>	Break long patterns so that they fit into RFCs. The resulting patterns might not always be valid XSD, so use with care.

YANG Output

Options for the *yang* output format:

<code>--yang-canonical</code>	Generate all statements in the canonical order.
<code>--yang-remove-unused-imports</code>	Remove unused import statements from the output.

YIN Output

Options for the *yin* output format:

<code>--yin-canonical</code>	Generate all statements in the canonical order.
<code>--yin-pretty-strings</code>	Pretty print strings, i.e. print with extra whitespace in the string. This is not strictly correct, since the whitespace is significant within the strings in XML, but the output is more readable.

YANG Extensions

This section describes XPath functions that can be used in "must", "when", or "path" expressions in YANG modules, in addition to the core XPath 1.0 functions.

pyang can be instructed to reject the usage of these functions with the parameter `--strict`.

Function: *node-set* **deref**(*node-set*)

The **deref** function follows the reference defined by the first node in document order in the argument *node-set*, and returns the nodes it refers to.

If the first argument node is an instance-identifier, the function returns a node-set that contains the single node that the instance identifier refers to, if it exists. If no such node exists, an empty node-set is returned.

If the first argument node is a leafref, the function returns a node-set that contains the nodes that the leafref refers to.

If the first argument node is of any other type, an empty node-set is returned.

The following example shows how a leafref can be written with and without the deref function:

```
/* without deref */

leaf my-ip {
  type leafref {
    path "/server/ip";
  }
}

leaf my-port {
  type leafref {
    path "/server[ip = current()../my-ip]/port";
  }
}

/* with deref */

leaf my-ip {
  type leafref {
    path "/server/ip";
  }
}

leaf my-port {
  type leafref {
    path "deref(..my-ip)../port";
  }
}
```

Example

The following example validates the standard YANG modules with derived types:

```
$ pyang ietf-yang-types.yang ietf-inet-types.yang
```

The following example converts the ietf-yang-types module into YIN:

```
$ pyang -f yin -o ietf-yang-types.yin ietf-yang-types.yang
```

The following example converts the ietf-netconf-monitoring module into a UML diagram:

```
$ pyang -f uml ietf-netconf-monitoring.yang > \
  ietf-netconf-monitoring.uml
$ java -jar plantuml.jar ietf-netconf-monitoring.uml
$ open img/ietf-netconf-monitoring.png
```

Environment Variables

pyang searches for referred modules in the colon (:) separated path defined by the environment variable \$YANG_MODPATH and in the directory \$YANG_INSTALL/yang/modules.

pyang searches for plugins in the colon (:) separated path defined by the environment variable \$PYANG_PLUGINDIR.

Bugs

- 1 The XPath arguments for the *must* and *when* statements are checked only for basic syntax errors.

NCS man-pages, Volume 3



Name

confd_lib — C library for connecting to NSO

LIBRARY

NSO Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to NSO. The documentation for the library is divided into several manual pages:

confd_lib_lib(3)	Common Library Functions
confd_lib_dp(3)	The Data Provider API
confd_lib_events(3)	The Event Notification API
confd_lib_ha(3)	The High Availability API
confd_lib_cdb(3)	The CDB API
confd_lib_maapi(3)	The Management Agent API

There is also a C header file associated with each of these manual pages:

#include <confd_lib.h>	Common type definitions and prototypes for the functions in the confd_lib_lib(3) manual page. Always needed.
#include <confd_dp.h>	Needed when functions in the confd_lib_dp(3) manual page are used.
#include <confd_events.h>	Needed when functions in the confd_lib_events(3) manual page are used.
#include <confd_ha.h>	Needed when functions in the confd_lib_ha(3) manual page are used.
#include <confd_cdb.h>	Needed when functions in the confd_lib_cdb(3) manual page are used.
#include <confd_maapi.h>	Needed when functions in the confd_lib_maapi(3) manual page are used.

For backwards compatibility, #include <confd.h> can also be used, and is equivalent to:

```
#include <confd_lib.h>
#include <confd_dp.h>
#include <confd_events.h>
#include <confd_ha.h>
```

SEE ALSO

The NSO User Guide



Name

confd_lib_cdb — library for connecting to NSO built-in XML database (CDB)

Synopsis

```
#include <confd_lib.h> #include <confd_cdb.h>

int cdb_connect(int sock, enum cdb_sock_type type, const struct
sockaddr* srv, int srv_sz);

int cdb_connect_name(int sock, enum cdb_sock_type type, const struct
sockaddr* srv, int srv_sz, const char *name);

int cdb_mandatory_subscriber(int sock, const char *name);

int cdb_set_namespace(int sock, int hashed_ns);

int cdb_end_session(int sock);

int cdb_start_session(int sock, enum cdb_db_type db);

int cdb_start_session2(int sock, enum cdb_db_type db, int flags);

int cdb_close(int sock);

int cdb_wait_start(int sock);

int cdb_get_phase(int sock, struct cdb_phase *phase);

int cdb_get_txid(int sock, struct cdb_txid *txid);

int cdb_initiate_journal_compaction(int sock);

int cdb_load_file(int sock, const char *filename, int flags);

int cdb_load_str(int sock, const char *xml_str, int flags);

int cdb_get_user_session(int sock);

int cdb_get_transaction_handle(int sock);

int cdb_set_timeout(int sock, int timeout_secs);

int cdb_exists(int sock, const char *fmt, ...);

int cdb_cd(int sock, const char *fmt, ...);

int cdb_pushd(int sock, const char *fmt, ...);

int cdb_popd(int sock);

int cdb_getcwd(int sock, size_t strsz, char *curdir);

int cdb_getcwd_kpath(int sock, confd_hkeypath_t **kp);
```

```

int cdb_num_instances(int sock, const char *fmt, ...);

int cdb_next_index(int sock, const char *fmt, ...);

int cdb_index(int sock, const char *fmt, ...);

int cdb_is_default(int sock, const char *fmt, ...);

int cdb_subscribe2(int sock, enum cdb_sub_type type, int flags, int
priority, int *spoint, int nspace, const char *fmt, ...);

int cdb_subscribe(int sock, int priority, int nspace, int *spoint,
const char *fmt, ...);

int cdb_oper_subscribe(int sock, int nspace, int *spoint, const char
*fmt, ...);

int cdb_subscribe_done(int sock);

int cdb_trigger_subscriptions(int sock, int sub_points[], int len);

int cdb_trigger_oper_subscriptions(int sock, int sub_points[], int len,
int flags);

int cdb_diff_match(int sock, int subid, struct xml_tag tags[], int
tagslen);

int cdb_read_subscription_socket(int sock, int sub_points[], int
*resultlen);

int cdb_read_subscription_socket2(int sock, enum cdb_sub_notification
*type, int *flags, int *subpoints[], int *resultlen);

int cdb_replay_subscriptions(int sock, struct cdb_txid *txid, int
sub_points[], int len);

int cdb_get_replay_txids(int sock, struct cdb_txid **txid, int
*resultlen);

int cdb_diff_iterate(int sock, int subid, enum cdb_iter_ret (*iter, )
(confd_hkeypath_t *kp, enum cdb_iter_op op, confd_value_t *oldv,
confd_value_t *newv, void *state), int flags, void *initstate);

int cdb_diff_iterate_resume(int sock, enum cdb_iter_ret reply,
enum cdb_iter_ret (*iter, )( confd_hkeypath_t *kp, enum cdb_iter_op
op, confd_value_t *oldv, confd_value_t *newv, void *state), void
*resumestate);

int cdb_cli_diff_iterate(int sock, int subid, cli_diff_iter_function_t
*iter, int flags, void *initstate);

int cdb_get_modifications(int sock, int subid, int flags,
confd_tag_value_t **values, int *nvalues, const char *fmt, ...);

int cdb_get_modifications_iter(int sock, int flags, confd_tag_value_t
**values, int *nvalues);

```

```

int cdb_get_modifications_cli(int sock, int subid, int flags, char
**str);

int cdb_sync_subscription_socket(int sock, enum
cdb_subscription_sync_type st);

int cdb_sub_progress(int sock, const char *fmt, ...);

int cdb_sub_abort_trans(int sock, enum confd_errcode code, u_int32_t
apptag_ns, u_int32_t apptag_tag, const char *fmt, ...);

int cdb_sub_abort_trans_info(int sock, enum confd_errcode code,
u_int32_t apptag_ns, u_int32_t apptag_tag, const confd_tag_value_t
*error_info, int n, const char *fmt, ...);

int cdb_get_case(int sock, const char *choice, confd_value_t *rcase,
const char *fmt, ...);

int cdb_get(int sock, confd_value_t *v, const char *fmt, ...);

int cdb_get_int8(int sock, int8_t *rval, const char *fmt, ...);

int cdb_get_int16(int sock, int16_t *rval, const char *fmt, ...);

int cdb_get_int32(int sock, int32_t *rval, const char *fmt, ...);

int cdb_get_int64(int sock, int64_t *rval, const char *fmt, ...);

int cdb_get_u_int8(int sock, u_int8_t *rval, const char *fmt, ...);

int cdb_get_u_int16(int sock, u_int16_t *rval, const char *fmt, ...);

int cdb_get_u_int32(int sock, u_int32_t *rval, const char *fmt, ...);

int cdb_get_u_int64(int sock, u_int64_t *rval, const char *fmt, ...);

int cdb_get_bit32(int sock, u_int32_t *rval, const char *fmt, ...);

int cdb_get_bit64(int sock, u_int64_t *rval, const char *fmt, ...);

int cdb_get_bitbig(int sock, unsigned char **rval, int *bufsiz, const
char *fmt, ...);

int cdb_get_ipv4(int sock, struct in_addr *rval, const char *fmt, ...);

int cdb_get_ipv6(int sock, struct in6_addr *rval, const char
*fmt, ...);

int cdb_get_double(int sock, double *rval, const char *fmt, ...);

int cdb_get_bool(int sock, int *rval, const char *fmt, ...);

int cdb_get_datetime(int sock, struct confd_datetime *rval, const char
*fmt, ...);

int cdb_get_date(int sock, struct confd_date *rval, const char
*fmt, ...);

```

```

int cdb_get_time(int sock, struct confd_time *rval, const char
*fmt, ...);

int cdb_get_duration(int sock, struct confd_duration *rval, const char
*fmt, ...);

int cdb_get_enum_value(int sock, int32_t *rval, const char *fmt, ...);

int cdb_get_objectref(int sock, confd_hkeypath_t **rval, const char
*fmt, ...);

int cdb_get_oid(int sock, struct confd_snmp_oid **rval, const char
*fmt, ...);

int cdb_get_buf(int sock, unsigned char **rval, int *bufsiz, const char
*fmt, ...);

int cdb_get_buf2(int sock, unsigned char *rval, int *n, const char
*fmt, ...);

int cdb_get_str(int sock, char *rval, int n, const char *fmt, ...);

int cdb_get_binary(int sock, unsigned char **rval, int *bufsiz, const
char *fmt, ...);

int cdb_get_hexstr(int sock, unsigned char **rval, int *bufsiz, const
char *fmt, ...);

int cdb_get_qname(int sock, unsigned char **prefix, int *prefixsz,
unsigned char **name, int *namesz, const char *fmt, ...);

int cdb_get_list(int sock, confd_value_t **values, int *n, const char
*fmt, ...);

int cdb_get_ipv4prefix(int sock, struct confd_ipv4_prefix *rval, const
char *fmt, ...);

int cdb_get_ipv6prefix(int sock, struct confd_ipv6_prefix *rval, const
char *fmt, ...);

int cdb_get_decimal64(int sock, struct confd_decimal64 *rval, const
char *fmt, ...);

int cdb_get_identityref(int sock, struct confd_identityref *rval, const
char *fmt, ...);

int cdb_get_ipv4_and_plen(int sock, struct confd_ipv4_prefix *rval,
const char *fmt, ...);

int cdb_get_ipv6_and_plen(int sock, struct confd_ipv6_prefix *rval,
const char *fmt, ...);

int cdb_get_dquad(int sock, struct confd_dotted_quad *rval, const char
*fmt, ...);

int cdb_vget(int sock, confd_value_t *v, const char *fmt, va_list
args);

```

```

int cdb_get_object(int sock, confd_value_t *values, int n, const char
*fmt, ...);

int cdb_get_objects(int sock, confd_value_t *values, int n, int ix, int
nobj, const char *fmt, ...);

int cdb_get_values(int sock, confd_tag_value_t *values, int n, const
char *fmt, ...);

int cdb_set_elem(int sock, confd_value_t *val, const char *fmt, ...);

int cdb_set_elem2(int sock, const char *strval, const char *fmt, ...);

int cdb_vset_elem(int sock, confd_value_t *val, const char *fmt,
va_list args);

int cdb_set_case(int sock, const char *choice, const char *scase, const
char *fmt, ...);

int cdb_create(int sock, const char *fmt, ...);

int cdb_delete(int sock, const char *fmt, ...);

int cdb_set_object(int sock, const confd_value_t *values, int n, const
char *fmt, ...);

int cdb_set_values(int sock, const confd_tag_value_t *values, int n,
const char *fmt, ...);

```

LIBRARY

NSO Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to the NSO built-in XML database, CDB. The purpose of this API is to provide a read and subscription API to CDB.

CDB owns and stores the configuration data and the user of the API wants to read that configuration data and also get notified when someone through either NETCONF, SNMP, the CLI, the Web UI or the MAAPI modifies the data so that the application can re-read the configuration data and act accordingly.

CDB can also store operational data, i.e. data which is designated with a "config false" statement in the YANG data model. Operational data can be both read and written by the applications, but NETCONF and the other northbound agents can only read the operational data.

PATHS

The majority of the functions described here take as their two last arguments a format string and a variable number of extra arguments as in: `char *fmt, ...`;

The *fmt* is a printf style format string which is used to format a path into the XML data tree. Assume the following YANG fragment:

```
container hosts {
```

```

list host {
  key name;
  leaf name {
    type string;
  }
  leaf domain {
    type string;
  }
  leaf defgw {
    type inet:ipv4-address;
  }
  container interfaces {
    list interface {
      key name;
      leaf name {
        type string;
      }
      leaf ip {
        type inet:ipv4-address;
      }
      leaf mask {
        type inet:ipv4-address;
      }
      leaf enabled {
        type boolean;
      }
    }
  }
}

```

Furthermore, assuming our database is populated with the following data.

```

<hosts xmlns="http://example.com/ns/hst/1.0">
  <host>
    <name>buzz</name>
    <domain>tail-f.com</domain>
    <defgw>192.168.1.1</defgw>
    <interfaces>
      <interface>
        <name>eth0</name>
        <ip>192.168.1.61</ip>
        <mask>255.255.255.0</mask>
        <enabled>true</enabled>
      </interface>
      <interface>
        <name>eth1</name>
        <ip>10.77.1.44</ip>
        <mask>255.255.0.0</mask>
        <enabled>false</enabled>
      </interface>
    </interfaces>
  </host>
</hosts>

```

The format path `/hosts/host{buzz}/defgw` refers to the leaf called `defgw` of the host whose key (name leaf) is `buzz`.

The format path `/hosts/host{buzz}/interfaces/interface{eth0}/ip` refers to the leaf called `ip` in the `eth0` interface of the host called `buzz`.

It is possible loop through all entries in a list as in:


```
n = cdb_num_instances(sock, "/hosts/host");
for (i=0; i<n; i++) {
    cdb_cd(sock, "/hosts/host[%d]", i)
    ....
}
```

Thus instead of an actually instantiated key inside a pair of curly braces {key}, we can use a temporary integer key inside a pair of brackets [n].

We can use the following modifiers:

- %d requiring an integer parameter (type int) to be substituted.
- %u requiring an unsigned integer parameter (type unsigned int) to be substituted.
- %s requiring a char* string parameter to be substituted.
- %ip4 requiring a struct in_addr* to be substituted.
- %ip6 requiring a struct in6_addr* to be substituted.
- %x requiring a confd_value_t* to be substituted.
- .*x requiring an array length and a confd_value_t* pointing to an array of values to be substituted.
- %h requiring a confd_hkeypath_t* to be substituted.
- .*h requiring a length and a confd_hkeypath_t* to be substituted.

Thus,

```
char *hname = "earth";
struct in_addr ip;
ip.s_addr = inet_addr("127.0.0.1");

cdb_cd(sock, "/hosts/host{%s}/bar{%ip4}", hname, &ip);
```

would change the current position to the path: "/hosts/host{earth}/bar{127.0.0.1}"

It is also possible to use the different '%' modifiers outside the curly braces, thus the above example could have been written as:

```
char *prefix = "/hosts/host";
cdb_cd(sock, "%s{%s}/bar{%ip4}", prefix, hname, &ip);
```

If an element has multiple keys, the keys must be space separated as in `cdb_cd("/bars/bar{%s %d}/item", str, i);`. However the '.*x' modifier is an exception to this rule, and it is especially useful when we have a number of key values that are unknown at compile time. If we have a list `foo` which is known to have two keys, and we have those keys in an array `key[]`, we can use `cdb_cd("/foo{%x %x}", &key[0], &key[1]);`. But if the number of keys is unknown at compile time (or if we just want a more compact code), we can instead use `cdb_cd("/foo{.*x}", n, key);` where `n` is the number of keys.

The '%h' and '.*h' modifiers can only be used at the beginning of a format path, as they expand to the absolute path corresponding to the `confd_hkeypath_t`. These modifiers are particularly useful with `cdb_diff_iterate()` (see below), or for MAAPI access in data provider callbacks (see [confd_lib_maapi\(3\)](#) and [confd_lib_dp\(3\)](#)). The '.*h' variant allows for using only the initial part of a `confd_hkeypath_t`, as specified by the preceding length argument (similar to '%.*s' for `printf(3)`).

For example, if the `iter()` function passed to `cdb_diff_iterate()` has been invoked with a `confd_hkeypath_t *kp` that corresponds to `/hosts/host{buzz}`, we can read the `defgw` child element with

```
confd_value_t v;
cdb_get(s, &v, "%h/defgw", kp);
```

or the entire list entry with

```
confd_value_t v[5];
cdb_get_object(sock, v, 5, "%h", kp);
```

or the defgw child element for host mars with

```
confd_value_t v;
cdb_get(s, &v, "%h{mars}/defgw", kp->len - 1, kp);
```

All the functions that take a path on this form also have a `va_list` variant, of the same form as `cdb_vget()` and `cdb_vset_elem()`, which are the only ones explicitly documented below. I.e. they have a prefix "cdb_v" instead of "cdb_", and take a single `va_list` argument instead of a variable number of arguments.

FUNCTIONS

All functions return `CONF_OK` (0), `CONF_ERR` (-1) or `CONF_EOF` (-2) unless otherwise stated. `CONF_EOF` means that the socket to NSO has been closed.

Whenever `CONF_ERR` is returned from any API function described here, it is possible to obtain additional information on the error through the symbol `confd_errno`, see the [ERRORS](#) section in the [confd_lib_lib\(3\)](#) manual page.

```
int cdb_connect(int sock, enum cdb_sock_type type, const struct
sockaddr* srv, int srv_sz);
```

The application has to connect to NSO before it can interact. There are two different types of connections identified by `cdb_sock_type`:

<code>CDB_DATA_SOCKET</code>	This is a socket which is used to read configuration data, or to read and write operational data.
<code>CDB_SUBSCRIPTION_SOCKET</code>	This is a socket which is used to receive notifications about updates to the database. A subscription socket needs to be part of the application poll set.

Additionally the type `CDB_READ_SOCKET` is accepted for backwards compatibility - it is equivalent to `CDB_DATA_SOCKET`.

A call to `cdb_connect()` is typically followed by a call to either `cdb_start_session()` for a reading session or a call to `cdb_subscribe()` for a subscription socket.



Note

If this call fails (i.e. does not return `CONF_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

Errors: `CONF_ERR_MALLOC`, `CONF_ERR_OS`

```
int cdb_connect_name(int sock, enum cdb_sock_type type, const struct
sockaddr* srv, int srv_sz, const char *name);
```

When we use `cdb_connect()` to create a connection to NSO/CDB, the `name` parameter passed to the library initialization function `confd_init()` (see [confd_lib_lib\(3\)](#)) is used to identify the connection in status reports and logs. If we want different names to be used for different connections from

the same application process, we can use `cdb_connect_name()` with the wanted name instead of `cdb_connect()`.



Note

If this call fails (i.e. does not return `CONFD_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int cdb_mandatory_subscriber(int sock, const char *name);
```

Attaches a mandatory attribute and a mandatory name to the subscriber identified by *sock*. The *name* parameter is distinct from the name parameter in `cdb_connect_name`.

CDB keeps a list of mandatory subscribers for infinite extent, i.e. until `confd` is restarted. The function is idempotent.

Absence of one or more mandatory subscribers will result in abort of all transactions. A mandatory subscriber must be present during the entire PREPARE delivery phase.

If a mandatory subscriber crashes during a PREPARE delivery phase, the subscriber should be restarted and the commit operation should be retried.

A mandatory subscriber is present if the subscriber has issued at least one `cdb_subscribe2()` call followed by a `cdb_subscribe_done()` call.

A call to `cdb_mandatory_subscriber()` is only allowed before the first call of `cdb_subscribe2()`.



Note

Only applicable for two-phase subscribers.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int cdb_set_namespace(int sock, int hashed_ns);
```

If we want to access data in CDB where the toplevel element name is not unique, we need to set the namespace. We are reading data related to a specific `.fxs` file. `confdc` can be used to generate a `.h` file with a `#define` for the namespace, by the flag `--emit-h` to `confdc` (see [confdc\(1\)](#)).

It is also possible to indicate which namespace to use through the namespace prefix when we read and write data. Thus the path `/foo:bar/baz` will get us `/bar/baz` in the namespace with prefix "foo" regardless of what the "set" namespace is. And if there is only one toplevel element called "bar" across all namespaces, we can use `/bar/baz` without the prefix and without calling `cdb_set_namespace()`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`

```
int cdb_end_session(int sock);
```

We use `cdb_connect()` to establish a read socket to CDB. When the socket is closed, the read session is ended. We can reuse the same socket for another read session, but we must then end the session and create another session using `cdb_start_session()`.

While we have a live CDB read session for configuration data, CDB is normally locked for writing. Thus all external entities trying to modify CDB are blocked as long as we have an open CDB read session. It is

very important that we remember to either `cdb_end_session()` or `cdb_close()` once we have read what we wish to read.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`

```
int cdb_start_session(int sock, enum cdb_db_type db);
```

Starts a new session on an already established socket to CDB. The `db` parameter should be one of:

<code>CDB_RUNNING</code>	Creates a read session towards the running database.
<code>CDB_PRE_COMMIT_RUNNING</code>	Creates a read session towards the running database as it was before the current transaction was committed. This is only possible between a subscription notification and the final <code>cdb_sync_subscription_socket()</code> . At any other time trying to call <code>cdb_start_session()</code> will fail with <code>confd_errno</code> set to <code>CONFD_ERR_NOEXISTS</code> .

In the case of a `CDB_SUB_PREPARE` subscription notification a session towards `CDB_PRE_COMMIT_RUNNING` will (in spite of the name) will return values as they were *before the transaction which is about to be committed* took place. This means that if you want to read the new values during a `CDB_SUB_PREPARE` subscription notification you need to create a session towards `CDB_RUNNING`. However, since it is locked the session needs to be started in lockless mode using `cdb_start_session2()`. So for example:

```
cdb_read_subscription_socket2(ss, &type, &flags, &subp, &len);
/* ... */
switch (type) {
case CDB_SUB_PREPARE:
    /* Set up a lockless session to read new values: */
    cdb_start_session2(s, CDB_RUNNING, 0);
    read_new_config(s);
    cdb_end_session(s);
    cdb_sync_subscription_socket(ss, CDB_DONE_PRIORITY);
    break;
/* ... */
```

`CDB_STARTUP`

Creates a read session towards the startup database.

`CDB_OPERATIONAL`

Creates a read/write session towards the operational database. For further details about working with operational data in CDB, see the `OPERATIONAL DATA` section below.



Note

Subscriptions on operational data will not be triggered from a session created with this function - to trigger operational data subscriptions, we need to use `cdb_start_session2()`, see below.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_LOCKED`, `CONFD_ERR_NOEXISTS`

If the error is `CONFD_ERR_LOCKED` it means that we are trying to create a new CDB read session precisely when the write phase of some transaction is occurring. Thus correct usage of `cdb_start_session()` is:

```
while (1) {
    if (cdb_start_session(sock, CDB_RUNNING) == CONFD_OK)
```

```

        break;
    if (confd_errno == CONFD_ERR_LOCKED) {
        sleep(1);
        continue;
    }
    .... handle error
}

```

Alternatively we can use `cdb_start_session2()` with `flags = CDB_LOCK_SESSION|CDB_LOCK_WAIT`. This means that the call will block until the lock has been acquired, and thus we do not need the retry loop.

```
int cdb_start_session2(int sock, enum cdb_db_type db, int flags);
```

This function may be used instead of `cdb_start_session()` if it is considered necessary to have more detailed control over some aspects of the CDB session - if in doubt, use `cdb_start_session()` instead. The `sock` and `db` arguments are the same as for `cdb_start_session()`, and these values can be used for `flags` (ORed together if more than one):

```

#define CDB_LOCK_WAIT      (1 << 0)
#define CDB_LOCK_SESSION  (1 << 1)
#define CDB_LOCK_REQUEST  (1 << 2)
#define CDB_LOCK_PARTIAL  (1 << 3)

```

The flags affect sessions for the different database types as follows:

CDB_RUNNING	CDB_LOCK_SESSION obtains a read lock for the complete session, i.e. using this flag alone is equivalent to calling <code>cdb_start_session()</code> . CDB_LOCK_REQUEST obtains a read lock only for the duration of each read request. This means that values of elements read in different requests may be inconsistent with each other, and the consequences of this must be carefully considered. In particular, the use of <code>cdb_num_instances()</code> and the <code>[n]</code> "integer index" notation in keypaths is inherently unsafe in this mode. Note: The implementation will not actually obtain a lock for a single-value request, since that is an atomic operation anyway. The CDB_LOCK_PARTIAL flag is not allowed.
CDB_STARTUP	Same as CDB_RUNNING.
CDB_PRE_COMMIT_RUNNING	This database type does not have any locks, which means that it is an error to call <code>cdb_start_session2()</code> with any CDB_LOCK_XXX flag included in <code>flags</code> . Using a <code>flags</code> value of 0 is equivalent to calling <code>cdb_start_session()</code> .
CDB_OPERATIONAL	CDB_LOCK_REQUEST obtains a "subscription lock" for the duration of each write request. This can be described as an "advisory exclusive" lock, i.e. only one client at a time can hold the lock (unless CDB_LOCK_PARTIAL is used), but the lock does not affect clients that do not attempt to obtain it. It also does not affect the reading of operational data. The purpose of this lock is to indicate that the client wants the write operation to generate subscription notifications. The lock remains in effect until any/all subscription notifications generated as a result of the write has been delivered. If the CDB_LOCK_PARTIAL flag is used together with CDB_LOCK_REQUEST, the "subscription lock" only applies to the smallest data subtree that includes all the data in the write request. This means that multiple writes that generates subscription notifications, and

delivery of the corresponding notifications, can proceed in parallel as long as they affect disjunct parts of the data tree.

The CDB_LOCK_SESSION flag is not allowed. Using a *flags* value of 0 is equivalent to calling `cdb_start_session()`.

In all cases of using CDB_LOCK_SESSION or CDB_LOCK_REQUEST described above, adding the CDB_LOCK_WAIT flag means that instead of failing with CONFD_ERR_LOCKED if the lock can not be obtained immediately, requests will wait for the lock to become available. When used with CDB_LOCK_SESSION it pertains to `cdb_start_session2()` itself, with CDB_LOCK_REQUEST it pertains to the individual requests.

While it is possible to use this function to start a session towards a configuration database type with no locking at all (*flags* = 0), this is strongly discouraged in general, since it means that even the values read in a single multi-value request (e.g. `cdb_get_object()`, see below) may be inconsistent with each other. However it is necessary to do this if we want to have a session open during semantic validation, see the "Semantic Validation" chapter in the User Guide - and in this particular case it is safe, since the transaction lock prevents changes to CDB during validation.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_LOCKED, CONFD_ERR_NOEXISTS, CONFD_ERR_PROTOUSAGE

```
int cdb_close(int sock);
```

Closes the socket. `cdb_end_session()` should be called before calling this function.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

Even if the call returns an error, the socket will be closed.

```
int cdb_wait_start(int sock);
```

This call waits until CDB has completed start-phase 1 and is available, when it is CONFD_OK is returned. If CDB already is available (i.e. start-phase >= 1) the call returns immediately. This can be used by a CDB client who is not synchronously started and only wants to wait until it can read its configuration. The call can be used after `cdb_connect()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int cdb_get_phase(int sock, struct cdb_phase *phase);
```

Returns the start-phase CDB is currently in, in the struct `cdb_phase` pointed to by the second argument. Also if CDB is in phase 0 and has initiated an init transaction (to load any init files) the flag CDB_FLAG_INIT is set in the flags field of struct `cdb_phase` and correspondingly if an upgrade session is started the CDB_FLAG_UPGRADE is set. The call can be used after `cdb_connect()` and returns CONFD_OK.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int cdb_initiate_journal_compaction(int sock);
```

Normally CDB handles journal compaction of the config datastore automatically. If this has been turned off (in the configuration file) then the A.cdb file will grow indefinitely unless this API function is called periodically to initiate compaction. This function initiates a compaction and returns immediately (if the datastore is locked, the compaction will be delayed, but eventually compaction will take place). Calling this function when journal compaction is configured to be automatic has no effect.

Errors: -

```
int cdb_get_txid(int sock, struct cdb_txid *txid);
```

Read the last transaction id from CDB. This function can be used if we are forced to reconnect to CDB, If the transaction id we read is identical to the last id we had prior to loosing the CDB sockets we don't have to reload our managed object data. See the User Guide for full explanation. Returns CONFD_OK on success and CONFD_ERR or CONFD_EOF on failure.

```
int cdb_get_replay_txids(int sock, struct cdb_txid **txid, int *resultlen);
```

When the subscriptionReplay functionality is enabled in confd.conf this function returns the list of available transactions that CDB can replay. The current transaction id will be the first in the list, the second at txid[1] and so on. The number of transactions is returned in *resultlen*. In case there are no replay transactions available (the feature isn't enabled or there hasn't been any transactions yet) only one (the current) transaction id is returned. It is up to the caller to *free()* *txid* when it is no longer needed.

```
int cdb_set_timeout(int sock, int timeout_secs);
```

A timeout for client actions can be specified via `/confdConfig/cdb/clientTimeout` in `confd.conf`, see the [confd.conf\(5\)](#) manual page. This function can be used to dynamically extend (or shorten) the timeout for the current action. Thus it is possible to configure a restrictive timeout in `confd.conf`, but still allow specific actions to have a longer execution time.

The function can be called either with a subscription socket during subscription delivery on that socket (including from the *iter()* function passed to *cdb_diff_iterate()*), or with a data socket that has an active session. The timeout is given in seconds from the point in time when the function is called.



Note

The timeout for subscription delivery is common for all the subscribers receiving notifications at a given priority. Thus calling the function during subscription delivery changes the timeout for all the subscribers that are currently processing notifications.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_PROTOUSAGE, CONFD_ERR_BADSTATE

```
int cdb_exists(int sock, const char *fmt, ...);
```

Leafs in the data model may be optional, and presence containers and list entries may or may not exist. This function checks whether a node exists in CDB. Returns 0 for false, 1 for true and CONFD_ERR or CONFD_EOF for errors.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```
int cdb_cd(int sock, const char *fmt, ...);
```

Changes the working directory according to the format path. Note that this function can not be used as an existence test.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```
int cdb_pushd(int sock, const char *fmt, ...);
```

Similar to `cdb_cd()` but pushes the previous current directory on a stack.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSTACK, CONFD_ERR_BADPATH

```
int cdb_popd(int sock);
```

Pops the top element from the directory stack and changes directory to previous directory.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSTACK

```
int cdb_getcwd(int sock, size_t strsz, char *curdir);
```

Returns the current position as previously set by `cdb_cd()`, `cdb_pushd()`, or `cdb_popd()` as a string path. Note that what is returned is a pretty-printed version of the internal representation of the current position, it will be the shortest unique way to print the path but it might not exactly match the string given to `cdb_cd()`. The buffer in `*curdir` will be NULL terminated, and no more characters than `strsz-1` will be written to it.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int cdb_getcwd_kpath(int sock, confd_hkeypath_t **kp);
```

Returns the current position like `cdb_getcwd()`, but as a pointer to a hashed keypath instead of as a string. The `hkeypath` is dynamically allocated, and may further contain dynamically allocated elements. The caller must free the allocated memory, easiest done by calling `confd_free_hkeypath()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int cdb_num_instances(int sock, const char *fmt, ...);
```

Returns the number of entries in a list. On error CONFD_ERR or CONFD_EOF is returned.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```
int cdb_next_index(int sock, const char *fmt, ...);
```

Given a path to a list entry `cdb_next_index()` returns the position (starting from 0) of the next entry (regardless of whether the path exists or not). When the list has multiple keys a `*` may be used for the last keys to make the path partially instantiated. For example if `/foo/bar` has three integer keys, the following pseudo code could be used to iterate over all entries with 42 as the first key:

```
/* find the first entry of /foo/bar with 42 as first key */
ix = cdb_next_index(sock, "/foo/bar{42 * *}");
for (; ix >= 0; ix++) {
    int32_t k1 = 0;
    cdb_get_int32(sock, &k1, "/foo/bar[%d]/key1", ix);
    if (k1 != 42) break;
    /* ... do something with /foo/bar[%d] ... */
}
```

If there is no next entry -1 is returned. It is not possible to use this function on an ordered-by user list. On error CONFD_ERR or CONFD_EOF is returned.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```
int cdb_index(int sock, const char *fmt, ...);
```


Given a path to a list entry `cdb_index()` returns its position (starting from 0). On error `CONFD_ERR` or `CONFD_EOF` is returned.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`

```
int cdb_is_default(int sock, const char *fmt, ...);
```

This function returns 1 for a leaf which has a default value defined in the data model when no value has been set, i.e. when the default value is in effect. It returns 0 for other existing leafs, and `CONFD_ERR` or `CONFD_EOF` for errors. There is normally no need to call this function, since CDB automatically provides the default value as needed when `cdb_get()` etc is called.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`

```
int cdb_subscribe(int sock, int priority, int nspace, int *spoint,  
const char *fmt, ...);
```

Sets up a CDB subscription so that we are notified when CDB configuration data changes. There can be multiple subscription points from different sources, that is a single client daemon can have many subscriptions and there can be many client daemons.

Each subscription point is defined through a path similar to the paths we use for read operations. We can subscribe either to specific leafs or entire subtrees. Subscribing to list entries can be done using fully qualified paths, or tagpaths to match multiple entries. A path which isn't a leaf element automatically matches the subtree below that path. When specifying keys to a list entry it is possible to use the wildcard character `*` which will match any key value.

When subscribing to a leaf with a `tailf:default-ref` statement, or to a subtree with elements that have `tailf:default-ref`, implicit subscriptions to the referred leafs are added. This means that a change in a referred leaf will generate a notification for the subscription that has referring leaf(s) - but currently such a change will not be reported by `cdb_diff_iterate()`. Thus to get the new "effective" value of a referring leaf in this case, it is necessary to either read the value of the leaf with e.g. `cdb_get()` - or to use a subscription that includes the referred leafs, and use `cdb_diff_iterate()` when a notification for that subscription is received.

Some examples

<code>/hosts</code>	Means that we subscribe to any changes in the subtree - rooted at <code>/hosts</code> . This includes additions or removals of <code>host</code> entries as well as changes to already existing <code>host</code> entries.
<code>/hosts/host{www}/ interfaces/ interface{eth0}/ip</code>	Means we are notified when <code>host www</code> changes its IP address on <code>eth0</code> .
<code>/hosts/host/interfaces/ interface/ip</code>	Means we are notified when any <code>host</code> changes any of its IP addresses.
<code>/hosts/host/interfaces</code>	Means we are notified when either an interface is added/removed or when an individual leaf element in an existing interface is changed.

The *priority* value is an integer. When CDB is changed, the change is performed inside a transaction. Either a **commit** operation from the CLI or a **candidate-commit** operation in NETCONF means that the running database is changed. These changes occur inside a `ConfD` transaction. CDB will handle the subscriptions in lock-step priority order. First all subscribers at the lowest priority are handled, once they all have replied and synchronized through calls to `cdb_sync_subscription_socket()` the next set

- at the next priority level is handled by CDB. Priority numbers are global, i.e. if there are multiple client daemons notifications will still be delivered in priority order per all subscriptions, not per daemon.

See `cdb_diff_iterate()` and `cdb_diff_match()` for ways of filtering subscription notifications and finding out what changed. The easiest way is though to not use either of the two above mentioned diff function but to solely rely on the positioning of the subscription points in the tree to figure out what changed.

`cdb_subscribe()` returns a *subscription point* in the return parameter *spoint*. This integer value is used to identify this particular subscription.

Because there can be many subscriptions on the same socket the client must notify ConfD when it is done subscribing and ready to receive notifications. This is done using `cdb_subscribe_done()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS

```
int cdb_oper_subscribe(int sock, int nspace, int *spoint, const char
*fmt, ...);
```

Sets up a CDB subscription for changes in the operational data base. Similar to the subscriptions for configuration data, we can be notified of changes to the operational data stored in CDB. Note that there are several differences from the subscriptions for configuration data:

- Notifications are only generated if the writer has taken a subscription lock, see `cdb_start_session2()` above.
- Priorities are not used for these notifications.
- It is not possible to receive the previous value for modified leafs in `cdb_diff_iterate()`.
- A special synchronization reply must be used when the notifications have been read (see `cdb_sync_subscription_socket()` below).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS

```
int cdb_subscribe2(int sock, enum cdb_sub_type type, int flags, int
priority, int *spoint, int nspace, const char *fmt, ...);
```

This function supersedes the current `cdb_subscribe()` and `cdb_oper_subscribe()` as well as makes it possible to use the new two phase subscription method. The `cdb_sub_type` is defined as:

```
enum cdb_sub_type {
    CDB_SUB_RUNNING = 1,
    CDB_SUB_RUNNING_TWOPHASE = 2,
    CDB_SUB_OPERATIONAL = 3
};
```

The CDB subscription type `CDB_SUB_RUNNING` is the same as `cdb_subscribe()`, `CDB_SUB_OPERATIONAL` is the same as `cdb_oper_subscribe()`, and `CDB_SUB_RUNNING_TWOPHASE` does a two phase subscription.

The flags argument should be set to 0, or a combination of:

`CDB_SUB_WANT_ABORT_ON_ABORT` Normally if a subscriber is the one to abort a transaction it will not receive an abort notification. This flags means that this subscriber wants an abort notification even if it was the one that called

`cdb_sub_abort_trans()`. This flag is only valid when the subscription type is `CDB_SUB_RUNNING_TWOPHASE`.

The two phase subscriptions work like this: A subscriber uses `cdb_subscribe2()` with the type set to `CDB_SUB_RUNNING_TWOPHASE` to register as many subscription points as required. The `cdb_subscribe_done()` function is used to indicate that no more subscription points will be registered on that particular socket. Only after `cdb_subscribe_done()` is called will subscription notifications be delivered.

Once a transaction enters prepare state all CDB two phase subscribers will be notified in priority order (lowest priority first, subscribers with the same priority is delivered in parallel). The `cdb_read_subscription_socket2()` function will set type to `CDB_SUB_PREPARE`. Once all subscribers have acknowledged the notification by using the function `cdb_sync_subscription_socket(CDB_DONE_PRIORITY)` they will subsequently be notified when the transaction is committed. The `CDB_SUB_COMMIT` notification is the same as the current subscription mechanism, so when a transaction is committed all subscribers will be notified (again in priority order).

When a transaction is aborted, delivery of any remaining `CDB_SUB_PREPARE` notifications is cancelled. The subscribers that had already been notified with `CDB_SUB_PREPARE` will be notified with `CDB_SUB_ABORT` (This notification will be done in reverse order of the `CDB_SUB_PREPARE` notification). The transaction could be aborted because one of the subscribers that received `CDB_SUB_PREPARE` called `cdb_sub_abort_trans()`, but it could also be caused for other reasons, for example another data provider (than CDB) can abort the transaction.



Note

Two phase subscriptions are not supported for NCS.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`

```
int cdb_subscribe_done(int sock);
```

When a client is done registering all its subscriptions on a particular subscription socket it must call `cdb_subscribe_done()`. No notifications will be delivered until then.

```
int cdb_trigger_subscriptions(int sock, int sub_points[], int len);
```

This function makes it possible to trigger CDB subscriptions for configuration data even though the configuration has not been modified. The caller will trigger all subscription points passed in the `sub_points` array (or all subscribers if the array is of zero length) in priority order, and the call will not return until the last subscriber has called `cdb_sync_subscription_socket()`.

The call is blocking and doesn't return until all subscribers have acknowledged the notification. That means that it is not possible to use `cdb_trigger_subscriptions()` in a cdb subscriber process (without forking a process or spawning a thread) since it would cause a deadlock.

The subscription notification generated by this "synthetic" trigger will seem like a regular subscription notification to a subscription client. As such, it is possible to use `cdb_diff_iterate()` to traverse the changeset. CDB will make up this changeset in which all leafs in the configuration will appear to be set, and all list entries and presence containers will appear as if they are created.

If the client is a two-phase subscriber, a prepare notification will first be delivered and if any client aborts this synthetic transaction further delivery of subscription notification is suspended and an error is

returned to the caller of `cdb_trigger_subscriptions()`. The error is the result of mapping the `CONFD_ERRCODE` as set by the aborting client as described for MAAPI in the [EXTENDED ERROR REPORTING](#) section in the [confd_lib_lib\(3\)](#) manpage. Note however that the configuration is still the way it is - so it is up to the caller of `cdb_trigger_subscriptions()` to take appropriate action (for example: raising an alarm, restarting a subsystem, or even rebooting the system).

If one or more subscription ids is passed in the `subids` array that are not valid, an error (`CONFD_ERR_PROTOUSAGE`) will be returned and no subscriptions will be triggered. If no subscription ids are passed this error can not occur (even if there aren't any subscribers).

```
int cdb_trigger_oper_subscriptions(int sock, int sub_points[], int len,
int flags);
```

This function works like `cdb_trigger_subscriptions()`, but for CDB subscriptions to operational data. The caller will trigger all subscription points passed in the `sub_points` array (or all operational data subscribers if the array is of zero length), and the call will not return until the last subscriber has called `cdb_sync_subscription_socket()`.

Since the generation of subscription notifications for operational data requires that the subscription lock is taken (see `cdb_start_session2()`), this function implicitly attempts to take a "global" subscription lock. If the subscription lock is already taken, the function will by default return `CONFD_ERR` with `confd_errno` set to `CONFD_ERR_LOCKED`. To instead have it wait until the lock becomes available, `CDB_LOCK_WAIT` can be passed for the `flags` parameter.

```
int cdb_replay_subscriptions(int sock, struct cdb_txid *txid, int
sub_points[], int len);
```

This function makes it possible to replay the subscription events for the last configuration change to some or all CDB subscribers. This call is useful in a number of recovery scenarios, where some CDB subscribers lost connection to ConfD before having received all the changes in a transaction. The replay functionality is only available if it has been enabled in `confd.conf`

The caller specifies the transaction id of the last transaction that the application has completely seen and acted on. This verifies that the application has only missed (part of) the last transaction. If a different (older) transaction ID is specified, an error is returned and no subscriptions will be triggered. If the transaction id is the latest transaction ID (i.e. the caller is already up to date) nothing is triggered and `CONFD_OK` is returned.

By calling this function, the caller will potentially trigger all subscription points passed in the `sub_points` array (or all subscribers if the array is of zero length). The subscriptions will be triggered in priority order, and the call will not return until the last subscriber has called `cdb_sync_subscription_socket()`.

The call is blocking and doesn't return until all subscribers have acknowledged the notification. That means that it is not possible to use `cdb_replay_subscriptions()` in a cdb subscriber process (without forking a process or spawning a thread) since it would cause a deadlock.

The subscription notification generated by this "synthetic" trigger will seem like a regular subscription notification to a subscription client. It is possible to use `cdb_diff_iterate()` to traverse the changeset.

If the client is a two-phase subscriber, a prepare notification will first be delivered and if any client aborts this synthetic transaction further delivery of subscription notification is suspended and an error is returned to the caller of `cdb_replay_subscriptions()`. The error is the result of mapping the `CONFD_ERRCODE` as set by the aborting client as described for MAAPI in the [EXTENDED ERROR REPORTING](#) section in the [confd_lib_lib\(3\)](#) manpage.

```
int cdb_read_subscription_socket(int sock, int sub_points[], int
*resultlen);
```

The subscription socket - which is acquired through a call to `cdb_connect()` - must be part of the application poll set. Once the subscription socket has I/O ready to read, we must call `cdb_read_subscription_socket()` on the subscription socket.

The call will fill in the result in the array `sub_points` with a list of integer values containing *subscription points* earlier acquired through calls to `cdb_subscribe()`. The global variable `cdb_active_subscriptions` can be read to find how many active subscriptions the application has. Make sure the `sub_points[]` array is at least this big, otherwise the confd library will write in unallocated memory.

The subscription points may be either for configuration data or operational data (if `cdb_oper_subscribe()` has been used on the same socket), but they will all be of the same "type" - i.e. a single call of the function will never deliver a mix of configuration and operational data subscription points.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int cdb_read_subscription_socket2(int sock, enum cdb_sub_notification
*type, int *flags, int *subpoints[], int *resultlen);
```

```
enum cdb_sub_notification {
    CDB_SUB_PREPARE = 1,
    CDB_SUB_COMMIT = 2,
    CDB_SUB_ABORT = 3,
    CDB_SUB_OPER = 4
};
```

This is another version of the `cdb_read_subscription_socket()` with two important differences:

- 1 In this version *subpoints* is allocated by the library, and it is up to the caller of this function to `free()` it when it is done.
- 2 It is possible to retrieve the type of the subscription notification via the *type* return parameter.

All parameters except *sock* are return parameters. It is legal to pass in *flags* and *type* as NULL pointers (in which case type and flags cannot be retrieved). *subpoints* is an array of integers, the length is indicated in *resultlen*, it is allocated by the library, and *must be freed by the caller*. The *type* parameter is what the subscriber uses to distinguish the different types of subscription notifications.

The *flags* return parameter can have the following bits set:

CDB_SUB_FLAG_IS_LAST	This bit is set when this notification is the last of its type for this subscription socket.
CDB_SUB_FLAG_HA_IS_SLAVE	This bit is set when NCS runs in HA mode, and the current HA mode is slave. I.e. it is a convenient way for the subscriber to know whether this node is in slave mode or not.
CDB_SUB_FLAG_TRIGGER	This bit is set when the cause of the subscription notification is that someone called <code>cdb_trigger_subscriptions()</code> .
CDB_SUB_FLAG_REVERT	If a confirming commit is aborted it will look to the CDB subscriber as if a transaction happened that is the reverse of what the original transaction was. This bit will be set when such a transaction is the cause of the notification. Note that for a two-phase subscriber both

	a prepare and a commit notification is delivered. However it is not possible to reply by calling <code>cdb_sub_abort_trans()</code> for the prepare notification in this case, instead the subscriber will have to take appropriate backup action if it needs to abort (for example: raise an alarm, restart, or even reboot the system).
<code>CDB_SUB_FLAG_HA_SYNC</code>	This bit is set when the cause of the subscription notification is initial synchronization of a HA slave from CDB on the master.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int cdb_diff_iterate(int sock, int subid, enum cdb_iter_ret (*iter, )
(confd_hkeypath_t *kp, enum cdb_iter_op op, confd_value_t *oldv,
confd_value_t *newv, void *state), int flags, void *initstate);
```

After reading the subscription socket the `cdb_diff_iterate()` function can be used to iterate over the changes made in CDB data that matched the particular subscription point given by `subid`.

The user defined function `iter()` will be called for each element that has been modified and matches the subscription. The `iter()` callback receives the `confd_hkeypath_t kp` which uniquely identifies which node in the data tree that is affected, the operation, and optionally the values it has before and after the transaction. The `op` parameter gives the modification as:

<code>MOP_CREATED</code>	The list entry, presence container, or leaf of type <code>empty</code> given by <code>kp</code> has been created.
<code>MOP_DELETED</code>	The list entry, presence container, or optional leaf given by <code>kp</code> has been deleted. If the subscription was triggered because an ancestor was deleted, the <code>iter()</code> function will not be called at all if the delete was above the subscription point. However if the flag <code>ITER_WANT_ANCESTOR_DELETE</code> is passed to <code>cdb_diff_iterate()</code> then deletes that trigger a descendant subscription will also generate a call to <code>iter()</code> , and in this case <code>kp</code> will be the path that was actually deleted.
<code>MOP_MODIFIED</code>	A descendant of the list entry given by <code>kp</code> has been modified.
<code>MOP_VALUE_SET</code>	The value of the leaf given by <code>kp</code> has been set to <code>newv</code> .
<code>MOP_MOVED_AFTER</code>	The list entry given by <code>kp</code> , in an ordered-by user list, has been moved. If <code>newv</code> is <code>NULL</code> , the entry has been moved first in the list, otherwise it has been moved after the entry given by <code>newv</code> . In this case <code>newv</code> is a pointer to an array of key values identifying an entry in the list. The array is terminated with an element that has type <code>C_NOEXISTS</code> .

By setting the `flags` parameter `ITER_WANT_REVERSE` two-phase subscribers may use this function to traverse the reverse changeset in case of `CDB_SUB_ABORT` notification. In this scenario a two-phase subscriber traverses the changes in the prepare phase (`CDB_SUB_PREPARE` notification) and if the transaction is aborted the subscriber may iterate the inverse to the changes during the abort phase (`CDB_SUB_PREPARE` notification).

For configuration subscriptions, the previous value of the node can also be passed to `iter()` if the `flags` parameter contains `ITER_WANT_PREV`, in which case `oldv` will be pointing to it (otherwise `NULL`). For operational data subscriptions, the `ITER_WANT_PREV` flag is ignored, and `oldv` is always `NULL` - there is no equivalent to `CDB_PRE_COMMIT_RUNNING` that holds "old" operational data.

If `iter()` returns `ITER_STOP`, no more iteration is done, and `CONFD_OK` is returned. If `iter()` returns `ITER_RECURSE` iteration continues with all children to the node. If `iter()` returns

ITER_CONTINUE iteration ignores the children to the node (if any), and continues with the node's sibling, and if `iter()` returns ITER_UP the iteration is continued with the node's parents sibling. If, for some reason, the `iter()` function wants to return control to the caller of `cdb_diff_iterate()` before all the changes has been iterated over it can return ITER_SUSPEND. The caller then has to call `cdb_diff_iterate_resume()` to continue/finish the iteration.

The *state* parameter can be used for any user supplied state (i.e. whatever is supplied as *initstate* is passed as *state* to `iter()` in each invocation).

By default the traverse order is undefined but guaranteed to be the most efficient one. The traverse order may be changed by setting setting a bit in the *flags* parameter:

ITER_WANT_SCHEMA_ORDER The `iter()` function will be invoked in *schema* order (i.e. in the order in which the elements are defined in the YANG file).

ITER_WANT_LEAF_FIRST_ORDER The `iter()` function will be invoked for leafs first, then non-leafs.

ITER_WANT_LEAF_LAST_ORDER The `iter()` function will be invoked for non-leafs first, then leafs.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS, CONFD_ERR_BADSTATE, CONFD_ERR_PROTOUSAGE.

```
int cdb_diff_iterate_resume(int sock, enum cdb_iter_ret reply,
enum cdb_iter_ret (*iter, )( confd_hkeypath_t *kp, enum cdb_iter_op
op, confd_value_t *oldv, confd_value_t *newv, void *state), void
*resumestate);
```

The application *must* call this function whenever an iterator function has returned ITER_SUSPEND to finish up the iteration. If the application does not wish to continue iteration it must at least call `cdb_diff_iterate_resume(s, ITER_STOP, NULL, NULL);` to clean up the state. The *reply* parameter is what the iterator function would have returned (i.e. normally ITER_RECURSE or ITER_CONTINUE) if it hadn't returned ITER_SUSPEND. Note that it is up to the iterator function to somehow communicate that it has returned ITER_SUSPEND to the caller of `cdb_diff_iterate()`, this can for example be a field in a struct for which a pointer to can passed back and forth in the *state/resumestate* variable.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS, CONFD_ERR_BADSTATE.

```
int cdb_diff_match(int sock, int subid, struct xml_tag tags[], int
tagslen);
```

This function can be invoked when a subscription point has fired. Similar to the `confd_hkp_tagmatch()` function it takes an argument which is an array of XML tags. The function will invoke `cdb_diff_iterate()` on a subscription socket. Using combinations of ITER_STOP, ITER_CONTINUE and ITER_RECURSE return values, the function checks a tagpath and decides whether any changes (under the subscription point) has occurred that also match the provided path *tags*. It is slightly easier to use this function than `cdb_diff_iterate()` but can also be slower since it is a general purpose matcher.

If we have a subscription point at `/root`, we could invoke this function as:

```
struct xml_tag tags[] = {{root_root, root_ns},
                        {root_servers, root_ns},
                        {root_server, root_ns}};
/* /root/servers/server */
int retv = cdb_diff_match(subsock, subpoint, tags, 3);
```

The function returns 1 if there were any changes under *subpoint* that matched *tags*, 0 if no match was found and CONFID_ERR on error.

```
int cdb_cli_diff_iterate(int sock, int subid, cli_diff_iter_function_t
*iter, int flags, void *initstate);
```

Where the cli_diff_iter_function_t is defined as:

```
typedef enum cdb_iter_ret
(cli_diff_iter_function_t)(confd_hkeypath_t *kp,
enum cdb_iter_op op,
confd_value_t *oldv,
confd_value_t *newv,
char *clistr,
int token_count,
struct confd_cli_token *tokens,
void *state);
```



Note

This function is DEPRECATED. Use cdb_get_modifications_cli() instead.

The function cdb_cli_diff_iterate() works just like the cdb_diff_iterate() function, except the iter() function takes three additional parameters, *clistr*, *token_count*, and *tokens*. The *clistr* is a string containing the (C-style) rendering of the CLI commands equivalent to the current keypath/operation. The *tokens* is actually an array of length *token_count*, it contains the CLI string broken down by token.

The string and the token array (including all the strings in the array) are allocated by the library, and will be freed by the library when the *iter* function returns.

If *flags* has the ITER_WANT_SCHEMA_ORDER bit set, then the iter() function will be invoked in *schema* order (i.e. in the order in which the elements are defined in the YANG file). Normally the order is undefined, which is most efficient.

Note that the cli commands are independent of whether the originating request actually came in over the CLI or some other northbound interface.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOEXISTS, CONFID_ERR_BADSTATE.

```
int cdb_get_modifications(int sock, int subid, int flags,
confd_tag_value_t **values, int *nvalues, const char *fmt, ...);
```

The cdb_get_modifications() function can be called after reception of a subscription notification to retrieve all the changes that caused the subscription notification. The socket *s* is the subscription socket, the subscription id must also be provided. Optionally a path can be used to limit what is returned further (only changes below the supplied path will be returned), if this isn't needed *fmt* can be set to NULL.

When cdb_get_modifications() returns CONFID_OK, the results are in *values*, which is a tag value array with length *nvalues*. The library allocates memory for the results, which must be freed by the caller. This can in all cases be done with code like this:

```
confd_tag_value_t *values;
int nvalues, i;

if (cdb_get_modifications(sock, subid, flags, &values, &nvalues,
```



```

        "/some/path") == CONFD_OK) {
    ...
    for (i = 0; i < nvalues; i++)
        confd_free_value(CONFD_GET_TAG_VALUE(&values[i]));
    free(values);
}

```

The tag value array differs somewhat between how it is described in the [confd_types\(3\)](#) manual page, most notably only the values that were modified in this transaction are included. In addition to that these are the different values of the tags depending on what happened in the transaction:

- A leaf of type empty that has been deleted has the value of C_NOEXISTS, and when it is created it has the value C_XMLTAG.
- A leaf or a leaf-list that has been set to a new value (or its default value) is included with that new value. If the leaf or leaf-list is optional, then when it is deleted the value is C_NOEXISTS.
- Presence containers are included when they are created or when they have modifications below them (by the usual C_XMLBEGIN, C_XMLEND pair). If a presence container has been deleted its tag is included, but has the value C_NOEXISTS.

By default `cdb_get_modifications()` does not include list instances (created, deleted, or modified) - but if the CDB_GET_MODS_INCLUDE_LISTS flag is included in the *flags* parameter, list instances will be included. Created and modified instances are included wrapped in the C_XMLBEGIN / C_XMLEND pair, with the keys first. Deleted list instances instead begin with C_XMLBEGINDEL, then follows the keys, immediately followed by a C_XMLEND.

If the CDB_GET_MODS_SUPPRESS_DEFAULTS flag is included in the *flags* parameter, a default value that comes into effect for a leaf due to an ancestor list entry or presence container being created will not be included, and a default value that comes into effect for a leaf due to a set value being deleted will be included as a deletion (i.e. with value C_NOEXISTS).

When processing a CDB_SUB_ABORT notification for a two phase subscription, it is also possible to request a list of "reverse" modifications instead of the normal "forward" list. This is done by including the CDB_GET_MODS_REVERSE flag in the *flags* parameter.

```

int cdb_get_modifications_iter(int sock, int flags, confd_tag_value_t
**values, int *nvalues);

```

The `cdb_get_modifications_iter()` is basically a convenient short-hand of the `cdb_get_modifications()` function intended to be used from within a iteration function started by `cdb_diff_iterate()`. In this case no subscription id is needed, and the path is implicitly the current position in the iteration.

Combining this call with `cdb_diff_iterate()` makes it for example possible to iterate over a list, and for each list instance fetch the changes using `cdb_get_modifications_iter()`, and then return ITER_CONTINUE to process next instance.



Note

Note: The CDB_GET_MODS_REVERSE flag is ignored by `cdb_get_modifications_iter()`. It will instead return a "forward" or "reverse" list of modifications for a CDB_SUB_ABORT notification according to whether the ITER_WANT_REVERSE flag was included in the *flags* parameter of the `cdb_diff_iterate()` call.

```

int cdb_get_modifications_cli(int sock, int subid, int flags, char
**str);

```

The `cdb_get_modifications_cli()` function can be called after reception of a subscription notification to retrieve all the changes that caused the subscription notification as a string in Cisco CLI format. The socket *s* is the subscription socket, the subscription id must also be provided. The *flags* parameter is currently unused, and should be set to zero for future compatibility.

The CLI string is `malloc(3)`ed by the library, and the caller must free the memory using `free(3)` when it is not needed any longer.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int cdb_sync_subscription_socket(int sock, enum
cdb_subscription_sync_type st);
```

Once we have read the subscription notification through a call to `cdb_read_subscription_socket()` and optionally used the `cdb_diff_iterate()` to iterate through the changes as well as acted on the changes to CDB, we must synchronize with CDB so that CDB can continue and deliver further subscription messages to subscribers with higher priority numbers.

There are four different types of synchronization replies the application can use in the enum `cdb_subscription_sync_type` parameter:

<code>CDB_DONE_PRIORITY</code>	This means that the application has acted on the subscription notification and CDB can continue to deliver further notifications.
<code>CDB_DONE_SOCKET</code>	This means that we are done. But regardless of priority, CDB shall not send any further notifications to us on our socket that are related to the currently executing transaction.
<code>CDB_DONE_TRANSACTION</code>	This means that CDB should not send any further notifications to any subscribers - including ourselves - related to the currently executing transaction.
<code>CDB_DONE_OPERATIONAL</code>	This should be used when a subscription notification for operational data has been read. It is the only type that should be used in this case, since the operational data does not have transactions and the notifications do not have priorities.

When using two phase subscriptions and `cdb_read_subscription_socket2()` has returned the type as `CDB_SUB_PREPARE` or `CDB_SUB_ABORT` the only valid response is `CDB_DONE_PRIORITY`.

For configuration data, the transaction that generated the subscription notifications is pending until all notifications have been acknowledged. A read lock on CDB is in effect while notifications are being delivered, preventing writes until delivery is complete.

For operational data, the writer that generated the subscription notifications is not directly affected, but the "subscription lock" remains in effect until all notifications have been acknowledged - thus subsequent attempts to obtain a "global" subscription lock, or a subscription lock using `CDB_LOCK_PARTIAL` for a non-disjunct subtree, will fail or block while notifications are being delivered (see `cdb_start_session2()` above). Write operations that don't attempt to obtain the subscription lock will proceed independent of the delivery of subscription notifications.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int cdb_sub_progress(int sock, const char *fmt, ...);
```

After receiving a subscription notification (using `cdb_read_subscription_socket()`) but before acknowledging it (or aborting, in the case of prepare subscriptions), it is possible to send progress reports

back to ConfD using the `cdb_sub_progress()` function. The socket `sock` must be the subscription socket, and it is allowed to call the function more than once to display more than one message. It is also possible to use this function in the diff-iterate callback function. A newline at the end of the string isn't necessary.

Depending on which north-bound interface that triggered the transaction, the string passed may be reported by that interface. Currently this is only presented in the CLI when the operator requests detailed reporting using the **commit | details** command.

Errors: CONF_D_ERR_MALLOC, CONF_D_ERR_OS

```
int cdb_sub_abort_trans(int sock, enum confd_errcode code, u_int32_t
apptag_ns, u_int32_t apptag_tag, const char *fmt, ...);
```

This function is to be called instead of `cdb_sync_subscription_socket()` when the subscriber wishes to abort the current transaction. It is only valid to call after `cdb_read_subscription_socket2()` has returned with type set to CDB_SUB_PREPARE. The arguments after `sock` are the same as to `confd_X_seterr_extended()` and give the caller a way of indicating the reason for the failure. Details can be found in the [EXTENDED ERROR REPORTING](#) section in the [confd_lib_lib\(3\)](#) manpage.

Errors: CONF_D_ERR_MALLOC, CONF_D_ERR_OS

```
int cdb_sub_abort_trans_info(int sock, enum confd_errcode code,
u_int32_t apptag_ns, u_int32_t apptag_tag, const confd_tag_value_t
*error_info, int n, const char *fmt, ...);
```

This function does the same as `cdb_sub_abort_trans()`, and additionally gives the possibility to provide contents for the NETCONF <error-info> element. See the [EXTENDED ERROR REPORTING](#) section in the [confd_lib_lib\(3\)](#) manpage.

Errors: CONF_D_ERR_MALLOC, CONF_D_ERR_OS

```
int cdb_get_user_session(int sock);
```

Returns the user session id for the transaction that triggered the current subscription notification. This function uses a subscription socket, and can only be called when a subscription notification for configuration data has been received on that socket, before `cdb_sync_subscription_socket()` has been called. Additionally, it is not possible to call this function from the `iter()` function passed to `cdb_diff_iterate()`. To retrieve full information about the user session, use `maapi_get_user_session()` (see [confd_lib_maapi\(3\)](#)).



Note

Note: When the ConfD High Availability functionality is used, the user session information is not available on slave nodes.

Errors: CONF_D_ERR_MALLOC, CONF_D_ERR_OS, CONF_D_ERR_BADSTATE, CONF_D_ERR_NOEXISTS

```
int cdb_get_transaction_handle(int sock);
```

Returns the transaction handle for the transaction that triggered the current subscription notification. This function uses a subscription socket, and can only be called when a subscription notification for configuration data has been received on that socket, before `cdb_sync_subscription_socket()`

has been called. Additionally, it is not possible to call this function from the `iter()` function passed to `cdb_diff_iterate()`.



Note

A CDB client is not expected to access the ConfD transaction store directly - this function should only be used for logging or debugging purposes.



Note

When the ConfD High Availability functionality is used, the transaction information is not available on slave nodes.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADSTATE, CONFID_ERR_NOEXISTS

```
int cdb_get(int sock, confd_value_t *v, const char *fmt, ...);
```

This function reads a value from the path in *fmt* and writes the result into the result parameter *confd_value_t*. The path must lead to a leaf element in the XML data tree. Note that for the *C_BUF*, *C_BINARY*, *C_LIST*, *C_OBJECTREF*, *C_OID*, *C_QNAME*, *C_HEXSTR*, and *C_BITBIG* *confd_value_t* types, the buffer(s) pointed to are allocated using `malloc(3)` - it is up to the user of this interface to free them using `confd_free_value()`.

Errors: CONFID_ERR_NOEXISTS, CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADPATH, CONFID_ERR_BADTYPE

All the type safe versions of `cdb_get()` described below, as well as `cdb_vget()`, also have the same possible Errors. When the type of the read value is wrong, `confd_errno` is set to `CONFID_ERR_BADTYPE` and the function returns `CONFID_ERR`. The YANG type is given in the descriptions below.

```
int cdb_get_int8(int sock, int8_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read int8 values.

```
int cdb_get_int16(int sock, int16_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read int16 values.

```
int cdb_get_int32(int sock, int32_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read int32 values.

```
int cdb_get_int64(int sock, int64_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read int64 values.

```
int cdb_get_u_int8(int sock, uint8_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read uint8 values.

```
int cdb_get_u_int16(int sock, uint16_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read uint16 values.

```
int cdb_get_u_int32(int sock, u_int32_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read uint32 values.

```
int cdb_get_u_int64(int sock, u_int64_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read uint64 values.

```
int cdb_get_bit32(int sock, u_int32_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read bits values where the highest assigned bit position for the type is 31.

```
int cdb_get_bit64(int sock, u_int64_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read bits values where the highest assigned bit position for the type is above 31 and below 64.

```
int cdb_get_bitbig(int sock, unsigned char **rval, int *bufsiz, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read bits values where the highest assigned bit position for the type is above 63. Upon successful return *rval* is pointing to a buffer of size *bufsiz*. It is up to the user of this function to free the buffer using `free(3)` when it is not needed any longer.

```
int cdb_get_ipv4(int sock, struct in_addr *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read inet:ipv4-address values.

```
int cdb_get_ipv6(int sock, struct in6_addr *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read inet:ipv6-address values.

```
int cdb_get_double(int sock, double *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read xs:float and xs:double values.

```
int cdb_get_bool(int sock, int *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read boolean values.

```
int cdb_get_datetime(int sock, struct confd_datetime *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read date-and-time values.

```
int cdb_get_date(int sock, struct confd_date *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read xs:date values.

```
int cdb_get_time(int sock, struct confd_time *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read `xs:time` values.

```
int cdb_get_duration(int sock, struct confd_duration *rval, const char
*fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read `xs:duration` values.

```
int cdb_get_enum_value(int sock, int32_t *rval, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read enumeration values. If we have:

```
typedef unboundedType {
    type enumeration {
        enum unbounded;
        enum infinity;
    }
}
```

The two enumeration values `unbounded` and `infinity` will occur as two `#define` integers in the `.h` file which is generated from the YANG module. Thus this function `cdb_get_enum_value()` populates an unsigned integer pointer.

```
int cdb_get_objectref(int sock, confd_hkeypath_t **rval, const char
*fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read instance-identifier values. Upon successful return `rval` is pointing to an allocated `confd_hkeypath_t`. It is up to the user of this function to free the `hkeypath` using `confd_free_hkeypath()` when it is not needed any longer.

```
int cdb_get_oid(int sock, struct confd_snmp_oid **rval, const char
*fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read object-identifier values. Upon successful return `rval` is pointing to an allocated `struct confd_snmp_oid`. It is up to the user of this function to free the struct using `free(3)` when it is not needed any longer.

```
int cdb_get_buf(int sock, unsigned char **rval, int *bufsiz, const char
*fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read string values. Upon successful return `rval` is pointing to a buffer of size `bufsiz`. It is up to the user of this function to free the buffer using `free(3)` when it is not needed any longer.

```
int cdb_get_buf2(int sock, unsigned char *rval, int *n, const char
*fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read string values. If the buffer returned by `cdb_get()` fits into `*n` bytes `CONF_OK` is returned and the buffer is copied into `*rval`. Upon successful return `*n` is set to the number of bytes copied into `*rval`.

```
int cdb_get_str(int sock, char *rval, int n, const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read string values. If the buffer returned by `cdb_get()` plus a terminating NUL fits into `n` bytes `CONF_OK` is returned and the buffer is copied into `*rval` (as well as a terminating NUL character).

```
int cdb_get_binary(int sock, unsigned char **rval, int bufsiz, const
char fmt, ...);
```

Type safe variant of `cdb_get()`, as `cdb_get_buf()` but for binary values. Upon successful return *rval* is pointing to a buffer of size *bufsiz*. It is up to the user of this function to free the buffer using `free(3)` when it is not needed any longer.

```
int cdb_get_hexstr(int sock, unsigned char **rval, int bufsiz, const
char fmt, ...);
```

Type safe variant of `cdb_get()`, as `cdb_get_buf()` but for yang:hex-string values. Upon successful return *rval* is pointing to a buffer of size *bufsiz*. It is up to the user of this function to free the buffer using `free(3)` when it is not needed any longer.

```
int cdb_get_qname(int sock, unsigned char **prefix, int prefixsz,
unsigned char **name, int namesz, const char fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read xs:QName values. Note that *prefixsz* can be zero (in which case *prefix* will be set to NULL). The space for prefix and name is allocated using `malloc()`, it is up to the user of this function to free them when no longer in use.

```
int cdb_get_list(int sock, confd_value_t **values, int n, const char
fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read values of a YANG leaf-list. The function will `malloc()` an array of `confd_value_t` elements for the list, and return a pointer to the array via the *values* parameter and the length of the array via the *n* parameter. The caller must free the memory for the values (see `cdb_get()`) and the array itself. An example that reads and prints the elements of a list of strings:

```
confd_value_t *values;
int i, n;

cdb_get_list(sock, &values, &n, "/system/cards");
for (i = 0; i < n; i++) {
    printf("card %d: %s\n", i, CONFD_GET_BUFPTR(&values[i]));
    confd_free_value(&values[i]);
}
free(values);
```

```
int cdb_get_ipv4prefix(int sock, struct confd_ipv4_prefix *rval, const
char fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read inet:ipv4-prefix values.

```
int cdb_get_ipv6prefix(int sock, struct confd_ipv6_prefix *rval, const
char fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read inet:ipv6-prefix values.

```
int cdb_get_decimal64(int sock, struct confd_decimal64 *rval, const
char fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read decimal64 values.

```
int cdb_get_identityref(int sock, struct confd_identityref *rval, const
char fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read identityref values.

```
int cdb_get_ipv4_and_plen(int sock, struct confd_ipv4_prefix *rval,
    const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read tailf:ipv4-address-and-prefix-length values.

```
int cdb_get_ipv6_and_plen(int sock, struct confd_ipv6_prefix *rval,
    const char *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read tailf:ipv6-address-and-prefix-length values.

```
int cdb_get_dquad(int sock, struct confd_dotted_quad *rval, const char
    *fmt, ...);
```

Type safe variant of `cdb_get()` which is used to read yang:dotted-quad values.

```
int cdb_vget(int sock, confd_value_t *v, const char *fmt, va_list
    args);
```

This function does the same as `cdb_get()`, but takes a single `va_list` argument instead of a variable number of arguments - i.e. similar to `vprintf()`. Corresponding `va_list` variants exist for all the functions that take a path as a variable number of arguments.

```
int cdb_get_object(int sock, confd_value_t *values, int n, const char
    *fmt, ...);
```

In some cases it can be motivated to read multiple values in one request - this will be more efficient since it only incurs a single round trip to ConfD, but usage is a bit more complex. This function reads at most *n* values from the container or list entry specified by the path, and places them in the *values* array, which is provided by the caller. The array is populated according to the specification of the *Value Array* format in the *XML STRUCTURES* section of the [confd_types\(3\)](#) manual page.

When reading from a container or list entry with mixed configuration and operational data (i.e. a config container or list entry that has some number of operational elements), some elements will have the "wrong" type - i.e. operational data in a session for CDB_RUNNING/CDB_STARTUP, or config data in a session for CDB_OPERATIONAL. Leaf elements of the "wrong" type will have a "value" of C_NOEXISTS in the array, while static or (existing) optional sub-container elements will have C_XMLTAG in all cases. Sub-containers or leafs provided by external data providers will always be represented with C_NOEXISTS, whether config or not.

On success, the function returns the actual number of elements in the container or list entry. I.e. if the return value is bigger than *n*, only the values for the first *n* elements are in the array, and the remaining values have been discarded. Note that given the specification of the array contents, there is always a fixed upper bound on the number of actual elements, and if there are no presence sub-containers, the number is constant.

As an example, with the YANG fragment in the [PATHS](#) section above, this code could be used to read the values for interface "eth0" on host "buzz":

```
char *path = "/hosts/host{buzz}/interfaces/interface{%s}";
confd_value_t v[4];
struct in_addr ip, mask;
int enabled;

cdb_get_object(sock, v, 4, path, "eth0");
```



```

/* v[0] is interface name, already known
   - must be freed since it's a C_BUF */
confd_free_value(&v[0]);
ip = CONFD_GET_IPV4(&v[1]);
mask = CONFD_GET_IPV4(&v[2]);
enabled = CONFD_GET_BOOL(&v[3]);

```

In this simple example, we assumed that the application was aware of the details of the data model, specifically that a `confd_value_t` array of length 4 would be sufficient for the values we wanted to retrieve, and at which positions in the array those values could be found. If we make use of schema information loaded from the ConfD daemon into the library (see [confd_types\(3\)](#)), we can avoid "hardwiring" these details. The following, more complex, example does the same as the above, but using only the names (in the form of `#defines` from the header file generated by `confdc --emit-h`) of the relevant leafs:

```

char *path = "/hosts/host{buzz}/interfaces/interface{%s}";
struct confd_cs_node *object = confd_cs_node_cd(NULL, path);
struct confd_cs_node *cur;
int n = confd_max_object_size(object);
int i;
confd_value_t v[n];
struct in_addr ip, mask;
int enabled;

cdb_get_object(sock, v, n, path, "eth0");
for (cur = object->children, i = 0;
     cur != NULL;
     cur = confd_next_object_node(object, cur, &v[i]), i++) {
    switch (cur->tag) {
    case hst_ip:
        ip = CONFD_GET_IPV4(&v[i]);
        break;
    case hst_mask:
        mask = CONFD_GET_IPV4(&v[i]);
        break;
    case hst_enabled:
        enabled = CONFD_GET_BOOL(&v[i]);
        break;
    }
    /* always free - it is a no-op if not needed */
    confd_free_value(&v[i]);
}

```

See [confd_lib_lib\(3\)](#) for the specification of the `confd_max_object_size()` and `confd_next_object_node()` functions. Also worth noting is that the return value from `confd_max_object_size()` is a constant for a given node in a given data model - thus we could optimize the above by calling `confd_max_object_size()` only at the first invocation of `cdb_get_object()` for a given node, making use of the `opaque` element of `struct confd_cs_node` to store the value:

```

char *path = "/hosts/host{buzz}/interfaces/interface{%s}";
struct confd_cs_node *object = confd_cs_node_cd(NULL, path);
int n;
struct in_addr ip, mask;
int enabled;

if (object->opaque == NULL) {
    n = confd_max_object_size(object);
    object->opaque = (void *)n;
} else {
    n = (int)object->opaque;
}

```

```

{
    struct confd_cs_node *cur;
    confd_value_t v[n];
    int i;

    cdb_get_object(sock, v, n, path, "eth0");
    for (cur = object->children, i = 0;
        cur != NULL;
        cur = confd_next_object_node(object, cur, &v[i]), i++) {
        ...
    }
}

```

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```

int cdb_get_objects(int sock, confd_value_t *values, int n, int ix, int
nobj, const char *fmt, ...);

```

Similar to `cdb_get_object()`, but reads multiple entries of a list based on the "instance integer" otherwise given within square brackets in the path - here the path must specify the list without the instance integer. At most *n* values from each of *nobj* entries, starting at entry *ix*, are read and placed in the *values* array.

The array must be at least $n * nobj$ elements long, and the values for list entry $ix + i$ start at element $array[i * n]$ (i.e. *ix* starts at $array[0]$, $ix+1$ at $array[n]$, and so on). On success, the highest actual number of values in any of the list entries read is returned. An error (CONFD_ERR_NOEXISTS) will be returned if we attempt to read more entries than actually exist (i.e. if $ix + nobj - 1$ is outside the range of actually existing list entries). Example - read the data for all interfaces on the host "buzz" (assuming that we have memory enough for that):

```

char *path = "/hosts/host{buzz}/interfaces/interface";
int n;

n = cdb_num_instances(sock, path);
{
    confd_value_t v[n*4];
    char name[n][64];
    struct in_addr ip[n], mask[n];
    int enabled[n];
    int i;

    cdb_get_objects(sock, v, 4, 0, n, path);
    for (i = 0; i < n*4; i += 4) {
        confd_pp_value(&name[i][0], 64, &v[i]);
        /* value must be freed since it's a C_BUF */
        confd_free_value(&v[i]);
        ip[i] = CONFD_GET_IPV4(&v[i+1]);
        mask[i] = CONFD_GET_IPV4(&v[i+2]);
        enabled[i] = CONFD_GET_BOOL(&v[i+3]);
    }

    /* configure interfaces... */
}

```

This simple example can of course be enhanced to use loaded schema information in a similar manner as for `cdb_get_object()` above.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS

```
int cdb_get_values(int sock, confd_tag_value_t *values, int n, const
char *fmt, ...);
```

Read an arbitrary set of sub-elements of a container or list entry. The *values* array must be pre-populated with *n* values based on the specification of the *Tagged Value Array* format in the *XML STRUCTURES* section of the [confd_types\(3\)](#) manual page, where the *confd_value_t* value element is given as follows:

- C_NOEXISTS means that the value should be read from CDB and stored in the array.
- C_PTR also means that the value should be read from CDB, but instead gives the expected type and a pointer to the type-specific variable where the value should be stored. Thus this gives a functionality similar to the type safe versions of *cdb_get()*.
- C_XMLBEGIN and C_XMLEND are used as per the specification.
- Key values to select list entries can be given with their values.
- As a special case, the "instance integer" can be used to select a list entry by using C_CDBBEGIN instead of C_XMLBEGIN (and no key values).



Note

When we use C_PTR, we need to take special care to free any allocated memory. When we use C_NOEXISTS and the value is stored in the array, we can just use *confd_free_value()* regardless of the type, since the *confd_value_t* has the type information. But with C_PTR, only the actual value is stored in the pointed-to variable, just as for *cdb_get_buf()*, *cdb_get_binary()*, etc, and we need to free the memory specifically allocated for the types listed in the description of *cdb_get()* above. See the corresponding *cdb_get_xxx()* functions for the details of how to do this.

All elements have the same position in the array after the call, in order to simplify extraction of the values - this means that optional elements that were requested but didn't exist will have C_NOEXISTS rather than being omitted from the array. However requesting a list entry that doesn't exist, or requesting non-CDB data, or operational vs config data, is an error. Note that when using C_PTR, the only indication of a non-existing value is that the destination variable has not been modified - it's up to the application to set it to some "impossible" value before the call when optional leafs are read.

In this rather complex example we first read only the "name" and "enabled" values for all interfaces, and then read "ip" and "mask" for those that were enabled - a total of two requests. Note that since the "interface" list begin/end elements are in the array, the path must not include the "interface" component. When reading values from a single container, it is generally simpler to have the container component (and keys or instance integer) in the path instead.

```
char *path = "/hosts/host{buzz}/interfaces";
int n = cdb_num_instances(sock, "%s/interface", path);
{
    /* when reading ip/mask, we need 5 elements per interface:
       begin + name (key) + ip + mask + end */
    confd_tag_value_t tv[n*5];
    char name[n][64];
    struct in_addr ip[n], mask[n];
    int i, j;
    int n_if;

    /* read name and enabled for all interfaces */
    j = 0;
    for (i = 0; i < n; i++) {
        CONFD_SET_TAG_CDBBEGIN(&tv[j], hst_interface, hst__ns, i); j++;
        CONFD_SET_TAG_NOEXISTS(&tv[j], hst_name); j++;
        CONFD_SET_TAG_NOEXISTS(&tv[j], hst_enabled); j++;
        CONFD_SET_TAG_XMLEND(&tv[j], hst_interface, hst__ns); j++;
    }
}
```

```

}
cdb_get_values(sock, tv, j, path);

/* extract name for enabled interfaces */
j = 0;
for (i = 0; i < n*4; i += 4) {
    int enabled = CONFD_GET_BOOL(CONFD_GET_TAG_VALUE(&tv[i+2]));
    confd_value_t *v = CONFD_GET_TAG_VALUE(&tv[i+1]);
    if (enabled) {
        confd_pp_value(&name[j][0], 64, v);
        j++;
    }
    /* name must be freed regardless since it's a C_BUF */
    confd_free_value(v);
}
n_if = j;

/* read ip and mask for enabled interfaces by key value (name) */
j = 0;
for (i = 0; i < n_if; i++) {
    CONFD_SET_TAG_XMLBEGIN(&tv[j], hst_interface, hst__ns);    j++;
    CONFD_SET_TAG_STR(&tv[j], hst_name, &name[i][0]);          j++;
    CONFD_SET_TAG_PTR(&tv[j], hst_ip, C_IPV4, &ip[i]);          j++;
    CONFD_SET_TAG_PTR(&tv[j], hst_mask, C_IPV4, &mask[i]);       j++;
    CONFD_SET_TAG_XMLEND(&tv[j], hst_interface, hst__ns);      j++;
}
cdb_get_values(sock, tv, j, path);

for (i = 0; i < n_if; i++) {
    /* configure interface i with ip[i] and mask[i]... */
}
}

```

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH,
CONFD_ERR_BADTYPE, CONFD_ERR_NOEXISTS

```

int cdb_get_case(int sock, const char *choice, confd_value_t *rcase,
const char *fmt, ...);

```

When we use the YANG choice statement in the data model, this function can be used to find the currently selected case, avoiding useless `cdb_get()` etc requests for elements that belong to other cases. The `fmt, ...` arguments give the path to the container or list entry where the choice is defined, and `choice` is the name of the choice. The case value is returned to the `confd_value_t` that `rcase` points to, as type `C_XMLTAG` - i.e. we can use the `CONFD_GET_XMLTAG()` macro to retrieve the hashed tag value. If no case is currently selected (i.e. for an optional choice that doesn't have a default case), the function will fail with `CONFD_ERR_NOEXISTS`.

If we have "nested" choices, i.e. multiple levels of choice statements without intervening container or list statements in the data model, the `choice` argument must give a '/'-separated path with alternating choice and case names, from the data node given by the `fmt, ...` arguments to the specific choice that the request pertains to.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH,
CONFD_ERR_NOEXISTS

OPERATIONAL DATA

It is possible for an application to store operational data (i.e. status and statistical information) in CDB, instead of providing it on demand via the callback interfaces described in the [confd_lib_dp\(3\)](#) manual

page. The operational database has no transactions and normally avoids the use of locks in order to provide light-weight access methods, however when the multi-value API functions below are used, all updates requested by a given function call are carried out atomically. Read about how to specify the storage of operational data in CDB via the `tailf:cdb-oper` extension in the [tailf_yang_extensions\(5\)](#) manual page.

To establish a session for operational data, the application needs to use `cdb_connect()` with `CDB_DATA_SOCKET` and `cdb_start_session()` with `CDB_OPERATIONAL`. After this, all the read and access functions above are available for use with operational data, and additionally the write functions described below. Configuration data can not be accessed in a session for operational data, nor vice versa - however it is possible to have both types of sessions active simultaneously on two different sockets, or to alternate the use of one socket via `cdb_end_session()`. The write functions can never be used in a session for configuration data.



Note

In order to trigger subscriptions on operational data, we must obtain a subscription lock via the use of `cdb_start_session2()` instead of `cdb_start_session()`, see above.

In YANG it is possible to define a list of operational data without any keys. For this type of list, we use a single "pseudo" key which is always of type `C_INT64` - see the [Operational Data chapter in the User Guide](#). This key isn't visible in the northbound agent interfaces, but is used in the functions described here just as if it was a "normal" key.

```
int cdb_set_elem(int sock, confd_value_t *val, const char *fmt, ...);

int cdb_set_elem2(int sock, const char *strval, const char *fmt, ...);
```

There are two different functions to set the value of a single leaf. The first takes the value from a `confd_value_t` struct, the second takes the string representation of the value.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`,
`CONFD_ERR_BADTYPE`, `CONFD_ERR_NOT_WRITABLE`

```
int cdb_vset_elem(int sock, confd_value_t *val, const char *fmt,
va_list args);
```

This function does the same as `cdb_set_elem()`, but takes a single `va_list` argument instead of a variable number of arguments - i.e. similar to `vprintf()`. Corresponding `va_list` variants exist for all the functions that take a path as a variable number of arguments.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`,
`CONFD_ERR_BADTYPE`, `CONFD_ERR_NOT_WRITABLE`

```
int cdb_create(int sock, const char *fmt, ...);
```

Create a new list entry, presence container, or leaf of type `empty`. Note that for list entries and containers, sub-elements will not exist until created or set via some of the other functions, thus doing implicit create via `cdb_set_object()` or `cdb_set_values()` may be preferred in this case.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`,
`CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_NOTCREATABLE`,
`CONFD_ERR_ALREADY_EXISTS`

```
int cdb_delete(int sock, const char *fmt, ...);
```

Delete a list entry, presence container, or leaf of type empty, and all its child elements (if any).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_NOT_WRITABLE, CONFD_ERR_NOTDELETABLE, CONFD_ERR_NOEXISTS

```
int cdb_set_object(int sock, const confd_value_t *values, int n, const char *fmt, ...);
```

Set all elements corresponding to the complete contents of a container or list entry, except for sub-lists. The *values* array must be populated with *n* values according to the specification of the *Value Array* format in the *XML STRUCTURES* section of the [confd_types\(3\)](#) manual page.

If the container or list entry itself, or any sub-elements that are specified as existing, do not exist before this call, they will be created, otherwise the existing values will be updated. Non-mandatory leafs and presence containers that are specified as not existing in the array, i.e. with value C_NOEXISTS, will be deleted if they existed before the call.

When writing to a container with mixed configuration and operational data (i.e. a config container or list entry that has some number of operational elements), all config leaf elements must be specified as C_NOEXISTS in the corresponding array elements, while config sub-container elements are specified with C_XMLTAG just as for operational data.

For a list entry, since the key elements must be present in the array, it is not required that the key values are included in the path given by *fmt*. If the key values *are* included in the path, the values of the key elements in the array are ignored.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_BADTYPE, CONFD_ERR_NOT_WRITABLE

```
int cdb_set_values(int sock, const confd_tag_value_t *values, int n, const char *fmt, ...);
```

Set arbitrary sub-elements of a container or list entry. The *values* array must be populated with *n* values according to the specification of the *Tagged Value Array* format in the *XML STRUCTURES* section of the [confd_types\(3\)](#) manual page.

If the container or list entry itself, or any sub-elements that are specified as existing, do not exist before this call, they will be created, otherwise the existing values will be updated. Both mandatory and optional elements may be omitted from the array, and all omitted elements are left unchanged. To actually delete a non-mandatory leaf or presence container as described for `cdb_set_object()`, it may (as an extension of the format) be specified as C_NOEXISTS instead of being omitted.

For a list entry, the key values can be specified either in the path or via key elements in the array - if the values are in the path, the key elements can be omitted from the array. For sub-lists present in the array, the key elements must of course always also be present though, immediately following the C_XMLBEGIN element and in the order defined by the data model. It is also possible to delete a list entry by using a C_XMLBEGINDEL element, followed by the keys in data model order, followed by a C_XMLEND element.

For a list without keys (see above), the "pseudo" key may (or in some cases must) be present in the array, but of course there is no tag value for it, since it isn't present in the data model. In this case we must use a tag value of 0, i.e. it can be set with code like:

```
confd_tag_value_t tv[7];  
  
CONFD_SET_TAG_INT64(&tv[1], 0, 42);
```

The same method is used when reading data from such a list with the `cdb_get_values()` function described above.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADPATH, CONFID_ERR_BADTYPE, CONFID_ERR_NOT_WRITABLE

```
int cdb_set_case(int sock, const char *choice, const char *scase, const char *fmt, ...);
```

When we use the YANG choice statement in the data model, this function can be used to select the current case. When configuration data is modified by northbound agents, the current case is implicitly selected (and elements for other cases potentially deleted) by the setting of elements in a choice. For operational data in CDB however, this is under direct control of the application, which needs to explicitly set the current case. Setting the case will also automatically delete elements belonging to other cases, but it is up to the application to not set any elements in the "wrong" case.

The *fmt, ...* arguments give the path to the container or list entry where the choice is defined, and *choice* and *scase* are the choice and case names. For an optional choice, it is possible to have no case at all selected. To indicate that the previously selected case should be deleted without selecting another case, we can pass NULL for the *scase* argument.

If we have "nested" choices, i.e. multiple levels of choice statements without intervening container or list statements in the data model, the *choice* argument must give a '/'-separated path with alternating choice and case names, from the data node given by the *fmt, ...* arguments to the specific choice that the request pertains to.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADPATH, CONFID_ERR_NOTDELETABLE

```
int cdb_load_file(int sock, const char *filename, int flags);
```

Load operational data from *filename* into CDB operational. The file must be in xml format, and *sock* must be connected to CDB operational (i.e. `cdb_start_session()` or `cdb_start_session2()` must have been called with CDB_OPERATIONAL). If the file contains config data, or operational data not residing in CDB, that data will be silently ignored. If the name of the file ends in .gz (or .Z) then the file is assumed to be gzipped, and will be uncompressed as it is loaded.



Note

If you use a relative pathname for *filename*, it is taken as relative to the working directory of the ConfD daemon, i.e. the directory where the daemon was started.

Note that there are no transactions in CDB operational, so there will not be any validation or transactional commit of the file. However the file will be completely parsed before CDB tries to set the values, with the result that any errors in the file will abort the operation without changing anything in CDB operational.

The *flags* parameter is currently not used, and should be set to 0.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_BADPATH, CONFID_ERR_EXTERNAL

```
int cdb_load_str(int sock, const char *xml_str, int flags);
```

Load operational data from the string *xml_str* into CDB operational. I.e. instead of having the xml data read from a file as for `cdb_load_file()`, it is passed as a string to the function. Besides this, the function works the same as `cdb_load_file()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOT_WRITABLE,
CONFD_ERR_EXTERNAL

SEE ALSO

`confd_lib(3)` - Confd lib

`confd_types(3)` - Confd C data types

The Confd User Guide

Name

confd_lib_dp — callback library for connecting data providers to ConfD

Synopsis

```
#include <confd_lib.h> #include <confd_dp.h>

struct confd_daemon_ctx *confd_init_daemon(const char *name);

int confd_set_daemon_flags(struct confd_daemon_ctx *dx, int flags);

void confd_release_daemon(struct confd_daemon_ctx *dx);

int confd_connect(struct confd_daemon_ctx *dx, int sock, enum
confd_sock_type type, const struct sockaddr *srv, int addrsz);

int confd_register_trans_cb(struct confd_daemon_ctx *dx, const struct
confd_trans_cbs *trans);

int confd_register_db_cb(struct confd_daemon_ctx *dx, const struct
confd_db_cbs *dbcbs);

int confd_register_range_data_cb(struct confd_daemon_ctx *dx, const
struct confd_data_cbs *data, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);

int confd_register_data_cb(struct confd_daemon_ctx *dx, const struct
confd_data_cbs *data);

int confd_register_usess_cb(struct confd_daemon_ctx *dx, const struct
confd_usess_cbs *ucb);

int ncs_register_service_cb(struct confd_daemon_ctx *dx, const struct
ncs_service_cbs *scb);

int ncs_register_nano_service_cb(struct confd_daemon_ctx *dx,
const char *component_type, const char *state, const struct
ncs_nano_service_cbs *scb);

int confd_register_done(struct confd_daemon_ctx *dx);

int confd_fd_ready(struct confd_daemon_ctx *dx, int fd);

void confd_trans_set_fd(struct confd_trans_ctx *tctx, int sock);

int confd_data_reply_value(struct confd_trans_ctx *tctx, const
confd_value_t *v);

int confd_data_reply_value_array(struct confd_trans_ctx *tctx, const
confd_value_t *vs, int n);

int confd_data_reply_tag_value_array(struct confd_trans_ctx *tctx,
const confd_tag_value_t *tvs, int n);
```

```

int confd_data_reply_next_key(struct confd_trans_ctx *tctx, const
confd_value_t *v, int num_vals_in_key, long next);

int confd_data_reply_not_found(struct confd_trans_ctx *tctx);

int confd_data_reply_found(struct confd_trans_ctx *tctx);

int confd_data_reply_next_object_array(struct confd_trans_ctx *tctx,
const confd_value_t *v, int n, long next);

int confd_data_reply_next_object_tag_value_array(struct confd_trans_ctx
*tctx, const confd_tag_value_t *tv, int n, long next);

int confd_data_reply_next_object_arrays(struct confd_trans_ctx *tctx,
const struct confd_next_object *obj, int nobj, int timeout_millisecs);

int confd_data_reply_next_object_tag_value_arrays(struct
confd_trans_ctx *tctx, const struct confd_tag_next_object *tobj, int
nobj, int timeout_millisecs);

int confd_data_reply_attrs(struct confd_trans_ctx *tctx, const
confd_attr_value_t *attrs, int num_attrs);

int ncs_service_reply_proplist(struct confd_trans_ctx *tctx, const
struct ncs_name_value *proplist, int num_props);

int ncs_nano_service_reply_proplist(struct confd_trans_ctx *tctx, const
struct ncs_name_value *proplist, int num_props);

int confd_delayed_reply_ok(struct confd_trans_ctx *tctx);

int confd_delayed_reply_error(struct confd_trans_ctx *tctx, const char
*errstr);

int confd_data_set_timeout(struct confd_trans_ctx *tctx, int
timeout_secs);

void confd_trans_seterr(struct confd_trans_ctx *tctx, const char
*fmt, ...);

void confd_trans_seterr_extended(struct confd_trans_ctx *tctx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);

int confd_trans_seterr_extended_info(struct confd_trans_ctx *tctx,
enum confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);

void confd_db_seterr(struct confd_db_ctx *dbx, const char *fmt, ...);

void confd_db_seterr_extended(struct confd_db_ctx *dbx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);

```

```

int confd_db_seterr_extended_info(struct confd_db_ctx *dbx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);

int confd_db_set_timeout(struct confd_db_ctx *dbx, int timeout_secs);

int confd_aaa_reload(const struct confd_trans_ctx *tctx);

int confd_install_crypto_keys(struct confd_daemon_ctx* dtx);

void confd_register_trans_validate_cb(struct confd_daemon_ctx *dx,
const struct confd_trans_validate_cbs *vcbs);

int confd_register_valpoint_cb(struct confd_daemon_ctx *dx, const
struct confd_valpoint_cb *vcb);

int confd_register_range_valpoint_cb(struct confd_daemon_ctx *dx,
struct confd_valpoint_cb *vcb, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);

int confd_delayed_reply_validation_warn(struct confd_trans_ctx *tctx);

int confd_register_action_cbs(struct confd_daemon_ctx *dx, const struct
confd_action_cbs *acb);

int confd_register_range_action_cbs(struct confd_daemon_ctx *dx,
const struct confd_action_cbs *acb, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);

void confd_action_set_fd(struct confd_user_info *uinfo, int sock);

void confd_action_seterr(struct confd_user_info *uinfo, const char
*fmt, ...);

void confd_action_seterr_extended(struct confd_user_info *uinfo, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);

int confd_action_seterr_extended_info(struct confd_user_info *uinfo,
enum confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);

int confd_action_reply_values(struct confd_user_info *uinfo,
confd_tag_value_t *values, int nvalues);

int confd_action_reply_command(struct confd_user_info *uinfo, char
**values, int nvalues);

int confd_action_reply_rewrite(struct confd_user_info *uinfo, char
**values, int nvalues, char **unhides, int nunhides);

int confd_action_reply_rewrite2(struct confd_user_info *uinfo,
char **values, int nvalues, char **unhides, int nunhides, struct
confd_rewrite_select **selects, int nselects);

```

```

int confd_action_reply_completion(struct confd_user_info *uinfo, struct
confd_completion_value *values, int nvalues);

int confd_action_reply_range_enum(struct confd_user_info *uinfo, char
**values, int keysize, int nkeys);

int confd_action_delayed_reply_ok(struct confd_user_info *uinfo);

int confd_action_delayed_reply_error(struct confd_user_info *uinfo,
const char *errstr);

int confd_action_set_timeout(struct confd_user_info *uinfo, int
timeout_secs);

int confd_register_notification_stream(struct confd_daemon_ctx
*dx, const struct confd_notification_stream_cbs *ncbs, struct
confd_notification_ctx **nctx);

int confd_notification_send(struct confd_notification_ctx *nctx, struct
confd_datetime *time, confd_tag_value_t *values, int nvalues);

int confd_notification_replay_complete(struct confd_notification_ctx
*nctx);

int confd_notification_replay_failed(struct confd_notification_ctx
*nctx);

int confd_notification_reply_log_times(struct confd_notification_ctx
*nctx, struct confd_datetime *creation, struct confd_datetime *aged);

void confd_notification_set_fd(struct confd_notification_ctx *nctx, int
fd);

void confd_notification_set_snmp_src_addr(struct confd_notification_ctx
*nctx, const struct confd_ip *src_addr);

int confd_notification_set_snmp_notify_name(struct
confd_notification_ctx *nctx, const char *notify_name);

void confd_notification_seterr(struct confd_notification_ctx *nctx,
const char *fmt, ...);

void confd_notification_seterr_extended(struct confd_notification_ctx
*nctx, enum confd_errcode code, u_int32_t apptag_ns, u_int32_t
apptag_tag, const char *fmt, ...);

int confd_notification_seterr_extended_info(struct
confd_notification_ctx *nctx, enum confd_errcode code, u_int32_t
apptag_ns, u_int32_t apptag_tag, confd_tag_value_t *error_info, int n,
const char *fmt, ...);

int confd_register_snmp_notification(struct confd_daemon_ctx *dx,
int fd, const char *notify_name, const char *ctx_name, struct
confd_notification_ctx **nctx);

```

```

int confd_notification_send_snmp(struct confd_notification_ctx *nctx,
const char *notification, struct confd_snmp_varbind *varbinds, int
num_vars);

int confd_register_notification_snmp_inform_cb(struct confd_daemon_ctx
*dx, const struct confd_notification_snmp_inform_cbs *cb);

int confd_notification_send_snmp_inform(struct confd_notification_ctx
*nctx, const char *notification, struct confd_snmp_varbind *varbinds,
int num_vars, const char *cb_id, int ref);

int confd_register_notification_sub_snmp_cb(struct confd_daemon_ctx
*dx, const struct confd_notification_sub_snmp_cb *cb);

int confd_notification_flush(struct confd_notification_ctx *nctx);

int confd_register_auth_cb(struct confd_daemon_ctx *dx, const struct
confd_auth_cb *acb);

void confd_auth_seterr(struct confd_auth_ctx *actx, const char
*fmt, ...);

int confd_register_authorization_cb(struct confd_daemon_ctx *dx, const
struct confd_authorization_cbs *acb);

int confd_access_reply_result(struct confd_authorization_ctx *actx, int
result);

int confd_authorization_set_timeout(struct confd_authorization_ctx
*actx, int timeout_secs);

int confd_register_error_cb(struct confd_daemon_ctx *dx, const struct
confd_error_cb *ecb);

void confd_error_seterr(struct confd_user_info *uinfo, const char
*fmt, ...);

```

LIBRARY

ConfD Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to the ConfD Data Provider API. The purpose of this API is to provide callback hooks so that user-written data providers can provide data stored externally to ConfD. ConfD needs this information in order to drive its northbound agents.

The library is also used to populate items in the data model which are not data or configuration items, such as statistics items from the device.

The library consists of a number of API functions whose purpose is to install different callback functions at different points in the data model tree which is the representation of the device configuration. Read more about callpoints in [tailf_yang_extensions\(5\)](#). Read more about how to use the library in the User Guide chapters on Operational data and External data.

FUNCTIONS

```
struct confd_daemon_ctx *confd_init_daemon(const char *name);
```

Initializes a new daemon context or returns NULL on failure. For most of the library functions described here a `daemon_ctx` is required, so we must create a daemon context before we can use them. The daemon context contains a `d_opaque` pointer which can be used by the application to pass application specific data into the callback functions.

The *name* parameter is used in various debug printouts and is also used to uniquely identify the daemon. The **confd --status** will use this name when indicating which callpoints are registered.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_PROTOUSAGE

```
int confd_set_daemon_flags(struct confd_daemon_ctx *dx, int flags);
```

This function modifies the API behaviour according to the flags ORed into the *flags* argument. It should be called immediately after creating the daemon context with `confd_init_daemon()`. The following flags are available:

CONFD_DAEMON_FLAG_STRINGSONLY

If this flag is used, the callback functions described below will only receive string values for all instances of `confd_value_t` (i.e. the type is always `C_BUF`). The callbacks must also give only string values in their reply functions. This feature can be useful for proxy-type applications that are unaware of the types of all elements, i.e. data model agnostic.

CONFD_DAEMON_FLAG_REG_REPLACE_DISCONNECT

By default, if one daemon replaces a callpoint registration made by another daemon, this is only logged, and no action is taken towards the daemon that has "lost" its registration. This can be useful in some scenarios, e.g. it is possible to have an "initial default" daemon providing "null" data for many callpoints, until the actual data provider daemons have registered. If a daemon uses the `CONFD_DAEMON_FLAG_REG_REPLACE_DISCONNECT` flag, it will instead be disconnected from ConfD if any of its registrations are replaced by another daemon, and can take action as appropriate.

CONFD_DAEMON_FLAG_NO_DEFAULTS

This flag tells ConfD that the daemon does not store default values. By default, ConfD assumes that the daemon doesn't know about default values, and thus whenever default values come into effect, ConfD will issue `set_elem()` callbacks to set those values, even if they have not actually been set by the northbound agent. Similarly `set_case()` will be issued with the default case for choices that have one.

When the `CONFD_DAEMON_FLAG_NO_DEFAULTS` flag is set, ConfD will only issue `set_elem()` callbacks when values have been explicitly set, and `set_case()` when a case has been selected by explicitly setting an element in the case. Specifically:

- When a list entry or presence container is created, there will be no callbacks for descendant leafs with default value, or descendant choices with default case, unless values have been explicitly set.
- When a leaf with a default value is deleted, a `remove()` callback will be issued instead of a `set_elem()` with the default value.
- When the current case in a choice with default case is deleted without another case being selected, the `set_case()` callback will be invoked with the case value given as NULL instead of the default case.

**Note**

A daemon that has the `CONF_D_AEMON_FLAG_NO_DEFAULTS` flag set *must* reply to `get_elem()` and the other callbacks that request leaf values with a value of type `C_DEFAULT`, rather than the actual default value, when the default value for a leaf is in effect. It *must* also reply to `get_case()` with `C_DEFAULT` when the default case is in effect.

```
void confd_release_daemon(struct confd_daemon_ctx *dx);
```

Returns all memory that has been allocated by `confd_init_daemon()` and other functions for the daemon context. The control socket as well as all the worker sockets must be closed by the application (before or after `confd_release_daemon()` has been called).

```
int confd_connect(struct confd_daemon_ctx *dx, int sock, enum
confd_sock_type type, const struct sockaddr *srv, int addrsz);
```

Connects to the ConfD daemon. The `dx` parameter is a daemon context acquired through a call to `confd_init_daemon()`.

There are two different types of connected sockets between an external daemon and ConfD.

- | | |
|-----------------------------|--|
| <code>CONTROL_SOCKET</code> | The first socket that is connected must always be a control socket. All requests from ConfD to create new transactions will arrive on the control socket, but it is also used for a number of other requests that are expected to complete quickly - the general rule is that all callbacks that do not have a corresponding <code>init()</code> callback are in fact control socket requests. There can only be one control socket for a given daemon context. |
| <code>WORKER_SOCKET</code> | We must always create at least one worker socket. All transaction, data, validation, and action callbacks, except the <code>init()</code> callbacks, use a worker socket. It is possible for a daemon to have multiple worker sockets, and the <code>init()</code> callback (see e.g. <code>confd_register_trans_cb()</code>) must indicate which worker socket should be used for the subsequent requests. This makes it possible for an application to be multi-threaded, where different threads can be used for different transactions. |

Returns `CONF_D_OK` when successful or `CONF_D_ERR` on connection error.

**Note**

All the callbacks that are invoked via these sockets are subject to timeouts configured in `confd.conf`, see [confd.conf\(5\)](#). The callbacks invoked via the control socket must generate a reply back to ConfD within the time configured for `/confdConfig/capi/newSessionTimeout`, the callbacks invoked via a worker socket within the time configured for `/confdConfig/capi/queryTimeout`. If either timeout is exceeded, the daemon will be considered dead, and ConfD will disconnect it by closing the control and worker sockets.

**Note**

If this call fails (i.e. does not return `CONF_D_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

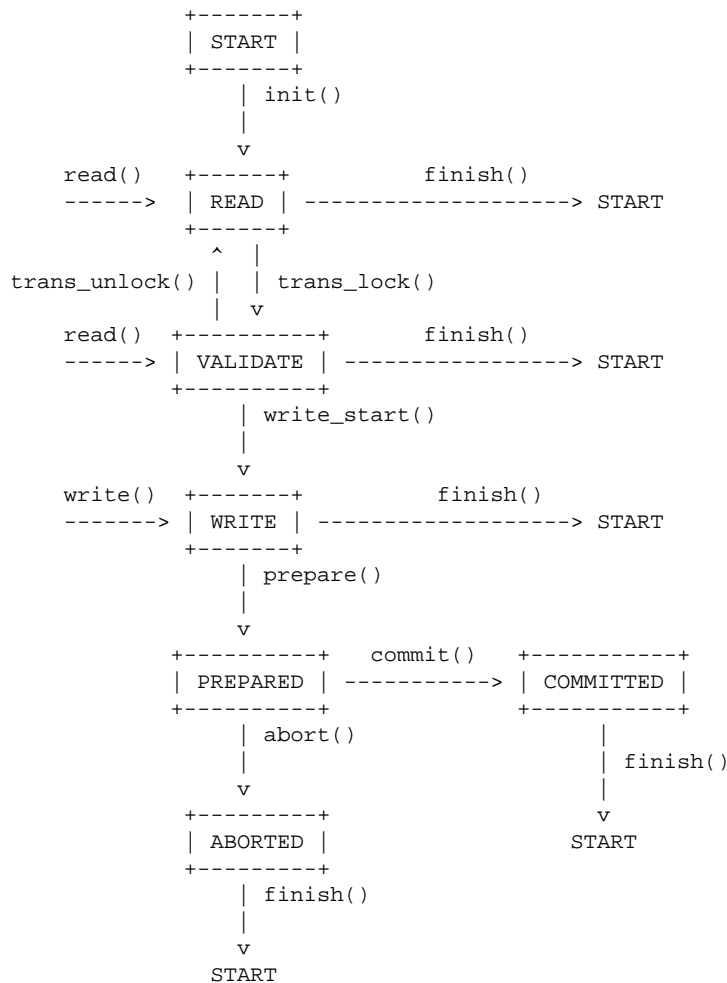
Errors: `CONF_D_ERR_MALLOC`, `CONF_D_ERR_OS`, `CONF_D_ERR_PROTOUSAGE`

```
int confd_register_trans_cb(struct confd_daemon_ctx *dx, const struct
confd_trans_cbs *trans);
```

This function registers transaction callback functions. A transaction is a ConfD concept. There may be multiple sources of data for the device configuration.

In order to orchestrate transactions with multiple sources of data, ConfD implements a two-phase commit protocol towards all data sources that participate in a transaction.

Each NETCONF operation will be an individual ConfD transaction. These transactions are typically very short lived. Transactions originating from the CLI or the Web UI have longer life. The ConfD transaction can be viewed as a conceptual state machine where the different phases of the transaction are different states and the invocations of the callback functions are state transitions. The following ASCII art depicts the state machine.



The struct confd_trans_cbs is defined as:

```
struct confd_trans_cbs {
    int (*init)(struct confd_trans_ctx *tctx);
    int (*trans_lock)(struct confd_trans_ctx *sctx);
    int (*trans_unlock)(struct confd_trans_ctx *sctx);
};
```



```

int (*write_start)(struct confd_trans_ctx *sctx);
int (*prepare)(struct confd_trans_ctx *tctx);
int (*abort)(struct confd_trans_ctx *tctx);
int (*commit)(struct confd_trans_ctx *tctx);
int (*finish)(struct confd_trans_ctx *tctx);
void (*interrupt)(struct confd_trans_ctx *tctx);
};

```

Transactions can be performed towards four different kinds of storages.

CONFID_CANDIDATE	If the system has been configured so that the external database owns the candidate data share, we will have to execute candidate transactions here. Usually ConfD owns the candidate and in that case the external database will never see any CONFID_CANDIDATE transactions.
CONFID_RUNNING	This is a transaction towards the actual running configuration of the device. All write operations in a CONFID_RUNNING transaction must be propagated to the individual subsystems that use this configuration data.
CONFID_STARTUP	If the system has been configured to support the NETCONF startup capability, this is a transaction towards the startup database.
CONFID_OPERATIONAL	This value indicates a transaction towards writable operational data. This transaction is used only if there are non-config data marked as <code>tailf:writable true</code> in the YANG module. Currently, these transactions are only started by the SNMP agent, and only when writable operational data is SET over SNMP.

Which type we have is indicated through the `confd_dbname` field in the `confd_trans_ctx`.

A transaction, regardless of whether it originates from the NETCONF agent, the CLI or the Web UI, has several distinct phases:

`init()`

This callback must always be implemented. All other callbacks are optional. This means that if the callback is set to NULL, ConfD will treat it as an implicit CONFID_OK. `libconfd` will allocate a transaction context on behalf of the transaction and give this newly allocated structure as an argument to the `init()` callback. The structure is defined as:

```

struct confd_user_info {
    int af; /* AF_INET | AF_INET6 */
    union {
        struct in_addr v4; /* address from where the */
        struct in6_addr v6; /* user session originates */
    } ip;
    u_int16_t port; /* source port */
    char username[MAXUSERNAMELEN]; /* who is the user */
    int usid; /* user session id */
    char context[MAXCTXLEN]; /* cli | webui | netconf | */
    /* noaaa | any MAAPI string */
    enum confd_proto proto; /* which protocol */
    struct confd_action_ctx actx; /* used during action call */
    time_t logintime;
    enum confd_usess_lock_mode lmode; /* the lock we have (only from */
    /* maapi_get_user_session()) */
    char snmp_v3_ctx[255]; /* SNMP context for SNMP sessions */
    /* empty string ("" ) for non-SNMP sessions */
    char clearpass[255]; /* if have the pass, it's here */
    /* only if confd internal ssh is used */
    int flags; /* CONFID_USESS_FLAG... */
};

```

```

    void *u_opaque;                /* Private User data */
    /* ConfD internal fields */
    char *errstr;                  /* for error formatting callback */
    int refc;
};

struct confd_trans_ctx {
    int fd;                        /* trans (worker) socket */
    int vfd;                       /* validation worker socket */
    struct confd_daemon_ctx *dx; /* our daemon ctx */
    enum confd_trans_mode mode;
    enum confd_dbname dbname;
    struct confd_user_info *uinfo;
    void *t_opaque;                /* Private User data (transaction) */
    void *v_opaque;                /* Private User data (validation) */
    struct confd_error error;      /* user settable via */
                                    /* confd_trans_seterr*() */
    struct confd_tr_item *accumulated;
    int thandle;                   /* transaction handle */
    void *cb_opaque;               /* private user data from */
                                    /* data callback registration */
    void *vcb_opaque;              /* private user data from */
                                    /* validation callback registration */
    int secondary_index;           /* if != 0: secondary index number */
                                    /* for list traversal */
    int validation_info;           /* CONFD_VALIDATION_FLAG_XXX */
    char *callpoint_opaque;        /* tailf:opaque for callpoint
                                    in data model */
    char *validate_opaque;         /* tailf:opaque for validation point
                                    in data model */
    union confd_request_data request_data; /* info from northbound agent */
    int hide_inactive;             /* if != 0: config data with
                                    CONFD_ATTR_INACTIVE should be hidden */

    /* ConfD internal fields */
    int index;                     /* array pos */
    int lastop;                   /* remember what we were doing */
    int last_proto_op;             /* ditto */
    int seen_reply;               /* have we seen a reply msg */
    int query_ref;                /* last query ref for this trans */
    int in_num_instances;
    u_int32_t num_instances;
    long nextarg;
    struct confd_data_cbs *next_dcb;
    confd_hkeypath_t *next_kp;
    struct confd_tr_item *lastack; /* tail of acklist */
    int refc;
};

```

This callback is required to prepare for future read/write operations towards the data source. It could be that a file handle or socket must be established. The place to do that is usually the `init()` callback.

The `init()` callback is conceptually invoked at the start of the transaction, but as an optimization, ConfD will as far as possible delay the actual invocation for a given daemon until it is required. In case of a read-only transaction, or a daemon that is only providing operational data, this can have the result that a daemon will not have any callbacks at all invoked (if none of the data elements that it provides are accessed).

The callback must also indicate to `libconfd` which `WORKER_SOCKET` should be used for future communications in this transaction. This is the mechanism which is used by `libconfd` to distribute

work among multiple worker threads in the database application. If another thread than the thread which owns the `CONTROL_SOCKET` should be used, it is up to the application to somehow notify that thread.

The choice of descriptor is done through the API call `confd_trans_set_fd()` which sets the `fd` field in the transaction context.

The callback must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE` or `CONFD_ERR`.

The transaction then enters `READ` state, where `ConfD` will perform a series of `read()` operations.

`trans_lock()`

This callback is invoked when the validation phase of the transaction starts. If the underlying database supports real transactions, it is usually appropriate to start such a native transaction here.

The callback must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE`, `CONFD_ERR`, or `CONFD_ALREADY_LOCKED`. The transaction enters `VALIDATE` state, where `ConfD` will perform a series of `read()` operations.

The trans lock is set until either `trans_unlock()` or `finish()` is called. `ConfD` ensures that a `trans_lock` is set on a single transaction only. In the case of the `CONFD_DELAYED_RESPONSE` - to later indicate that the database is already locked, use the `confd_delayed_reply_error()` function with the special error string "locked". An alternate way to indicate that the database is already locked is to use `confd_trans_seterr_extended()` (see below) with `CONFD_ERRCODE_IN_USE` - this is the only way to give a message in the "delayed" case. If this function is used, the callback must return `CONFD_ERR` in the "normal" case, and in the "delayed" case `confd_delayed_reply_error()` must be called with a `NULL` argument after `confd_trans_seterr_extended()`.

`trans_unlock()`

This callback is called when the validation of the transaction failed, or the validation is triggered explicitly (i.e. not part of a 'commit' operation). This is common in the CLI and the Web UI where the user can enter invalid data. Transactions that originate from `NETCONF` will never trigger this callback. If the underlying database supports real transactions and they are used, the transaction should be aborted here.

The callback must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE` or `CONFD_ERR`. The transaction re-enters `READ` state.

`write_start()`

This callback is invoked when the validation succeeded and the write phase of the transaction starts. If the underlying database supports real transactions, it is usually appropriate to start such a native transaction here.

The transaction enters the `WRITE` state. No more `read()` operations will be performed by `ConfD`.

The callback must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE`, `CONFD_ERR`, or `CONFD_IN_USE`.

If `CONFD_IN_USE` is returned, the transaction is restarted, i.e. it effectively returns to the `READ` state. To give this return code after `CONFD_DELAYED_RESPONSE`, use the `confd_delayed_reply_error()` function with the special error string "in_use". An alternative for both cases is to use `confd_trans_seterr_extended()` (see below) with `CONFD_ERRCODE_IN_USE` - this is the only way to give a message in the "delayed" case. If this function is used, the callback must return `CONFD_ERR` in the "normal" case, and in the "delayed" case `confd_delayed_reply_error()` must be called with a `NULL` argument after `confd_trans_seterr_extended()`.

`prepare()`

If we have multiple sources of data it is highly recommended that the callback is implemented. The callback is called at the end of the transaction, when all read and write operations for the transaction have been performed and the transaction should prepare to commit.

This callback should allocate the resources necessary for the commit, if any. The callback must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE`, `CONFD_ERR`, or `CONFD_IN_USE`.

If `CONFD_IN_USE` is returned, the transaction is restarted, i.e. it effectively returns to the `READ` state. To give this return code after `CONFD_DELAYED_RESPONSE`, use the `confd_delayed_reply_error()` function with the special error string "in_use". An alternative for both cases is to use `confd_trans_seterr_extended()` (see below) with `CONFD_ERRCODE_IN_USE` - this is the only way to give a message in the "delayed" case.

If this function is used, the callback must return `CONFD_ERR` in the "normal" case, and in the "delayed" case `confd_delayed_reply_error()` must be called with a `NULL` argument after `confd_trans_seterr_extended()`.

`commit()`

This callback is optional. This callback is responsible for writing the data to persistent storage. Must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE` or `CONFD_ERR`.

`abort()`

This callback is optional. This callback is responsible for undoing whatever was done in the `prepare()` phase. Must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE` or `CONFD_ERR`.

`finish()`

This callback is optional. This callback is responsible for releasing resources allocated in the `init()` phase. In particular, if the application choose to use the `t_opaque` field in the `confd_trans_ctx` to hold any resources, these resources must be released here.

`interrupt()`

This callback is optional. Unlike the other transaction callbacks, it does not imply a change of the transaction state, it is instead a notification that the user running the transaction requested that it should be interrupted (e.g. Ctrl-C in the CLI). Also unlike the other transaction callbacks, the callback request is sent asynchronously on the control socket. Registering this callback may be useful for a configuration data provider that has some (transaction or data) callbacks which require extensive processing - the callback could then determine whether one of these callbacks is being processed, and if feasible return an error from that callback instead of completing the processing. In that case, `confd_trans_seterr_extended()` with `code` `CONFD_ERRCODE_INTERRUPT` should be used.

All the callback functions (except `interrupt()`) must return `CONFD_OK`, `CONFD_DELAYED_RESPONSE` or `CONFD_ERR`.

It is often useful to associate an error string with a `CONFD_ERR` return value. This can be done through a call to `confd_trans_seterr()` or `confd_trans_seterr_extended()`.

Depending on the situation (original caller) the error string gets propagated to the CLI, the Web UI or the NETCONF manager.

```
int confd_register_db_cb(struct confd_daemon_ctx *dx, const struct
confd_db_cbs *dbcbs);
```

We may also optionally have a set of callback functions which span over several ConfD transactions.

If the system is configured in such a way so that the external database owns the candidate data store we must implement four callback functions to do this. If ConfD owns the candidate the candidate callbacks should be set to `NULL`.

If ConfD owns the candidate, and ConfD has been configured to support `confirmed-commit`, then three checkpointing functions must be implemented; otherwise these should be set to `NULL`. When `confirmed-commit` is enabled, the user can commit the candidate with a timeout. Unless a confirming commit is given by the user before the timer expires, the system must rollback to the previous running configuration. This mechanism is controlled by the checkpoint callbacks. See further below.

An external database may also (optionally) support the `lock/unlock` and `lock_partial/unlock_partial` operations. This is only interesting if there exists additional locking mechanisms towards the database - such as an external CLI which can lock the database, or if the external database owns the candidate.

Finally, the external database may optionally validate a candidate configuration. Configuration validation is preferably done through ConfD - however if a system already has implemented extensive configuration validation - the `candidate_validate()` callback can be used.

The struct `confd_db_cbs` structure looks like:

```
struct confd_db_cbs {
    int (*candidate_commit)(struct confd_db_ctx *dbx, int timeout);
    int (*candidate_confirming_commit)(struct confd_db_ctx *dbx);
    int (*candidate_reset)(struct confd_db_ctx *dbx);
    int (*candidate_chk_not_modified)(struct confd_db_ctx *dbx);
    int (*candidate_rollback_running)(struct confd_db_ctx *dbx);
    int (*candidate_validate)(struct confd_db_ctx *dbx);
    int (*add_checkpoint_running)(struct confd_db_ctx *dbx);
    int (*del_checkpoint_running)(struct confd_db_ctx *dbx);
    int (*activate_checkpoint_running)(struct confd_db_ctx *dbx);
    int (*copy_running_to_startup)(struct confd_db_ctx *dbx);
    int (*running_chk_not_modified)(struct confd_db_ctx *dbx);
    int (*lock)(struct confd_db_ctx *dbx, enum confd_dbname dbname);
    int (*unlock)(struct confd_db_ctx *dbx, enum confd_dbname dbname);
    int (*lock_partial)(struct confd_db_ctx *dbx,
                       enum confd_dbname dbname, int lockid,
                       confd_hkeypath_t paths[], int npaths);
    int (*unlock_partial)(struct confd_db_ctx *dbx,
                        enum confd_dbname dbname, int lockid);
    int (*delete_config)(struct confd_db_ctx *dbx,
                       enum confd_dbname dbname);
};
```

If we have an externally implemented candidate, that is if `confd.conf` item `/confdConfig/datastores/candidate/implementation` is set to "external", we must implement the 5 candidate callbacks. Otherwise (recommended) they must be set to `NULL`.

If implementation is "external", all databases (if there are more than one) MUST take care of the candidate for their part of the configuration data tree. If ConfD is configured to use an external database for parts of the configuration, and the built-in CDB database is used for some parts, CDB will handle the candidate for its part. See also `misc/extern_candidate` in the examples collection.

The callback functions are the following:

`candidate_commit()`

This function should copy the candidate DB into the running DB. If `timeout != 0`, we should be prepared to do a rollback or act on a `candidate_confirming_commit()`. The `timeout` parameter can not be used to set a timer for when to rollback; this timer is handled by the ConfD daemon. If we terminate without having acted on the `candidate_confirming_commit()`, we MUST restart with a rollback. Thus we must remember that we are waiting for a `candidate_confirming_commit()` and we must do so on persistent storage. Must only be implemented when the external database owns the candidate.

`candidate_confirming_commit()`

If the *timeout* in the `candidate_commit()` function is $\neq 0$, we will be either invoked here or in the `candidate_rollback_running()` function within *timeout* seconds. `candidate_confirming_commit()` should make the commit persistent, whereas a call to `candidate_rollback_running()` would copy back the previous running configuration to running.

`candidate_rollback_running()`

If for some reason, apart from a timeout, something goes wrong, we get invoked in the `candidate_rollback_running()` function. The function should copy back the previous running configuration to running.

`candidate_reset()`

This function is intended to copy the current running configuration into the candidate. It is invoked whenever the NETCONF operation `<discard-changes>` is executed or when a lock is released without committing.

`candidate_chk_not_modified()`

This function should check to see if the candidate has been modified or not. Returns `CONFD_OK` if no modifications has been done since the last commit or reset, and `CONFD_ERR` if any uncommitted modifications exist.

`candidate_validate()`

This callback is optional. If implemented, the task of the callback is to validate the candidate configuration. Note that the running database can be validated by the database in the `prepare()` callback. `candidate_validate()` is only meaningful when an explicit validate operation is received, e.g. through NETCONF.

`add_checkpoint_running()`

This function should be implemented only when ConfD owns the candidate, and confirmed-commit is enabled.

It is responsible for creating a checkpoint of the current running configuration and storing the checkpoint in non-volatile memory. When the system restarts this function should check if there is a checkpoint available, and use the checkpoint instead of running.

`del_checkpoint_running()`

This function should delete a checkpoint created by `add_checkpoint_running()`. It is called by ConfD when a confirming commit is received.

`activate_checkpoint_running()`

This function should rollback running to the checkpoint created by `add_checkpoint_running()`. It is called by ConfD when the timer expires or if the user session expires.

`copy_running_to_startup()`

This function should copy running to startup. It only needs to be implemented if the startup data store is enabled.

`running_chk_not_modified()`

This function should check to see if running has been modified or not. It only needs to be implemented if the startup data store is enabled. Returns `CONFD_OK` if no modifications have been done since the last copy of running to startup, and `CONFD_ERR` if any modifications exist.

`lock()`

This should only be implemented if our database supports locking from other sources than through ConfD. In this case both the `lock/unlock` and `lock_partial/unlock_partial` callbacks must be implemented. If a lock on the whole database is set through e.g. NETCONF, ConfD will first make sure that no other ConfD transaction has locked the database. Then it will call `lock()` to make sure

that the database is not locked by some other source (such as a non-ConfD CLI). Returns CONFD_OK on success, and CONFD_ERR if the lock was already held by an external entity.

`unlock()`

Unlocks the database.

`lock_partial()`

This should only be implemented if our database supports locking from other sources than through ConfD, see `lock()` above. This callback is invoked if a northbound agent requests a partial lock. The `paths[]` argument is an *npaths* long array of hkeypaths that identify the leafs and/or subtrees that are to be locked. The `lockid` is a reference that will be used on a subsequent corresponding `unlock_partial()` invocation.

`unlock_partial()`

Unlocks the partial lock that was requested with `lockid`.

`delete_config()`

Will be called for 'startup' or 'candidate' only. The database is supposed to be set to erased.

All the above callback functions must return either CONFD_OK or CONFD_ERR. If the system is configured so that ConfD owns the candidate, then obviously the candidate related functions need not be implemented. If the system is configured to not do confirmed commit, `candidate_confirming_commit()` and `candidate_commit()` need not to be implemented.

It is often interesting to associate an error string with a CONFD_ERR return value. In particular the `validate()` callback must typically indicate which item was invalid and why. This can be done through a call to `confd_db_seterr()` or `confd_db_seterr_extended()`.

Depending on the situation (original caller) the error string is propagated to the CLI, the Web UI or the NETCONF manager.

```
int confd_register_data_cb(struct confd_daemon_ctx *dx, const struct
confd_data_cbs *data);
```

This function registers the data manipulation callbacks. The data model defines a number of "callpoints". Each callpoint must have an associated set of data callbacks.

Thus if our database application serves three different callpoints in the data model we must install three different sets of data manipulation callbacks - one set at each callpoint.

The data callbacks either return data back to ConfD or they do not. For example the `create()` callback does not return data whereas the `get_next()` callback does. All the callbacks that return data do so through API functions, not by means of return values from the function itself.

The struct `confd_data_cbs` is defined as:

```
struct confd_data_cbs {
    char callpoint[MAX_CALLPOINT_LEN];
    /* where in the XML tree do we */
    /* want this struct */

    /* Only necessary to have this cb if our data model has */
    /* typeless optional nodes or oper data lists w/o keys */
    int (*exists_optional)(struct confd_trans_ctx *tctx,
                           confd_hkeypath_t *kp);
    int (*get_elem)(struct confd_trans_ctx *tctx,
                   confd_hkeypath_t *kp);
    int (*get_next)(struct confd_trans_ctx *tctx,
```

```

        confd_hkeypath_t *kp, long next);
int (*set_elem)(struct confd_trans_ctx *tctx,
                confd_hkeypath_t *kp,
                confd_value_t *newval);
int (*create)(struct confd_trans_ctx *tctx,
              confd_hkeypath_t *kp);
int (*remove)(struct confd_trans_ctx *tctx,
              confd_hkeypath_t *kp);
/* optional (find list entry by key/index values) */
int (*find_next)(struct confd_trans_ctx *tctx,
                 confd_hkeypath_t *kp,
                 enum confd_find_next_type type,
                 confd_value_t *keys, int nkeys);
/* optional optimizations */
int (*num_instances)(struct confd_trans_ctx *tctx,
                     confd_hkeypath_t *kp);
int (*get_object)(struct confd_trans_ctx *tctx,
                  confd_hkeypath_t *kp);
int (*get_next_object)(struct confd_trans_ctx *tctx,
                       confd_hkeypath_t *kp, long next);
int (*find_next_object)(struct confd_trans_ctx *tctx,
                        confd_hkeypath_t *kp,
                        enum confd_find_next_type type,
                        confd_value_t *keys, int nkeys);
/* next two are only necessary if 'choice' is used */
int (*get_case)(struct confd_trans_ctx *tctx,
                confd_hkeypath_t *kp, confd_value_t *choice);
int (*set_case)(struct confd_trans_ctx *tctx,
                confd_hkeypath_t *kp, confd_value_t *choice,
                confd_value_t *caseval);
/* next two are only necessary for config data providers,
   and only if /confdConfig/enableAttributes is 'true' */
int (*get_attr)(struct confd_trans_ctx *tctx,
                confd_hkeypath_t *kp,
                u_int32_t *attrs, int num_attrs);
int (*set_attr)(struct confd_trans_ctx *tctx,
                confd_hkeypath_t *kp,
                u_int32_t attr, confd_value_t *v);
/* only necessary if "ordered-by user" is used */
int (*move_after)(struct confd_trans_ctx *tctx,
                  confd_hkeypath_t *kp, confd_value_t *prevkeys);
/* only for per-transaction-invoked transaction hook */
int (*write_all)(struct confd_trans_ctx *tctx,
                 confd_hkeypath_t *kp);
void *cb_opaque; /* private user data */
};

```

One of the parameters to the callback is a `confd_hkeypath_t` (h - as in hashed keypath). This is fully described in [confd_types\(3\)](#).

The `cb_opaque` element can be used to pass arbitrary data to the callbacks, e.g. when the same set of callbacks is used for multiple callpoints. It is made available to the callbacks via an element with the same name in the transaction context (`tctx` argument), see the structure definition above.

If the `tailf:opaque` substatement has been used with the `tailf:callpoint` statement in the data model, the argument string is made available to the callbacks via the `callpoint_opaque` element in the transaction context.

When use of the `CONFID_ATTR_INACTIVE` attribute is enabled in the ConfD configuration (`/confdConfig/enableAttributes` and `/confdConfig/enableInactive` both set to `true`), read callbacks (`get_elem()` etc) for configuration data must observe the current value of the

hide_inactive element in the transaction context. If it is non-zero, those callbacks must act as if data with the CONFID_ATTR_INACTIVE attribute set does not exist.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_PROTOUSAGE

get_elem()

This callback function needs to return the value of a specific leaf. Assuming we have the following data model:

```
container servers {
    tailf:callpoint mycp;
    list server {
        key name;
        max-elements 64;
        leaf name {
            type string;
        }
        leaf ip {
            type inet:ip-address;
        }
        leaf port {
            type inet:port-number;
        }
    }
}
```

For example the value of the ip leaf in the server entry whose key is "www" can be returned separately. The way to return a single data item is through confd_data_reply_value().

The callback must return CONFID_OK on success, CONFID_ERR on error or CONFID_DELAYED_RESPONSE if the reply value is not yet available. In the latter case the application must at a later stage call confd_data_reply_value() (or confd_delayed_reply_ok() for a write operation). If an error is discovered at the time of a delayed reply, the error is signaled through a call to confd_delayed_reply_error()

If the leaf does not exist the callback must call confd_data_reply_not_found(). If the leaf has a default value defined in the data model, and no value has been set, the callback should use confd_data_reply_value() with a value of type C_DEFAULT - this makes it possible for northbound agents to leave such leaves out of the data returned to the user/manager (if requested).

The implementation of get_elem() must be prepared to return values for all the leaves including the key(s). When ConfD invokes get_elem() on a key leaf it is an existence test. The application should verify whether the object exists or not.

get_next()

This callback makes it possible for ConfD to traverse a set of list entries. The next parameter will be -1 on the first invocation. This function should reply by means of the function confd_data_reply_next_key().

If the list has a tailf:secondary-index statement (see [tailf_yang_extensions\(5\)](#)), and the entries are supposed to be retrieved according to one of the secondary indexes, the variable tctx->secondary_index will be set to a value greater than 0, indicating which secondary-index is used. The first secondary-index in the definition is identified with the value 1, the second with 2, and so on. confdc can be used to generate #defines for the index names. If no secondary indexes are defined, or if the sort order should be according to the key values, tctx->secondary_index is 0.

To signal that no more entries exist, we reply with a NULL pointer as the key value in the confd_data_reply_next_key() function.

The callback must return `CONF_OK` on success, `CONF_ERR` on error or `CONF_DELAYED_RESPONSE` if the reply value is not yet available. In the latter case the application must at a later stage call `conf_data_reply_next_key()`.



Note

For a list that does not specify a non-default sort order by means of a `ordered-by user` or `tailf:sort-order` statement, ConfD assumes that list entries are ordered strictly by increasing key (or secondary index) values. Thus for correct operation, we must observe this order when returning list entries in a sequence of `get_next()` calls.

`set_elem()`

This callback writes the value of a leaf. Note that an optional leaf with a type other than empty is created by a call to this function. The callback must return `CONF_OK` on success, `CONF_ERR` on error or `CONF_DELAYED_RESPONSE`.

`create()`

This callback creates a new list entry, a presence container, or a leaf of type empty. In the case of the `servers` data model above, this function need to create a new `server` entry. Must return `CONF_OK` on success, `CONF_ERR` on error, `CONF_DELAYED_RESPONSE` or `CONF_ACCUMULATE`.

The data provider is responsible for maintaining the order of list entries. If the list is marked as `ordered-by user` in the YANG data model, the `create()` callback must add the list entry to the end of the list.

`remove()`

This callback is used to remove an existing list entry, a presence container, or an optional leaf and all its sub nodes (if any). When we use the YANG `choice` statement in the data model, it may also be used to remove nodes that are not optional as such when a different case (or none) is selected. I.e. it must always be possible to remove cases in a choice.

Must return `CONF_OK` on success, `CONF_ERR` on error, `CONF_DELAYED_RESPONSE` or `CONF_ACCUMULATE`.

`exists_optional()`

If we have presence containers or leaves of type empty, we cannot use the `get_elem()` callback to read the value of such a node, since it does not have a type. An example of a data model could be:

```
container bs {
  presence "";
  tailf:callpoint bcp;
  list b {
    key name;
    max-elements 64;
    leaf name {
      type string;
    }
    container opt {
      presence "";
      leaf ii {
        type int32;
      }
    }
    leaf foo {
      type empty;
    }
  }
}
```

```
}
```

The above YANG fragment has 3 nodes that may or may not exist and that do not have a type. If we do not have any such elements, nor any operational data lists without keys (see below), we do not need to implement the `exists_optional()` callback and can set it to NULL.

If we have the above data model, we must implement the `exists_optional()`, and our implementation must be prepared to reply on calls of the function for the paths `/bs`, `/bs/b/opt`, and `/bs/b/foo`. The leaf `/bs/b/opt/ii` is not mandatory, but it does have a type namely `int32`, and thus the existence of that leaf will be determined through a call to the `get_elem()` callback.

The `exists_optional()` callback may also be invoked by ConfD as "existence test" for an entry in an operational data list without keys (see the [Operational Data chapter in the User Guide](#)). Normally this existence test is done with a `get_elem()` request for the first key, but since there are no keys, this callback is used instead. Thus if we have such lists, we must also implement this callback, and handle a request where the keypath identifies a list entry.

The callback must reply to ConfD using either the `confd_data_reply_not_found()` or the `confd_data_reply_found()` function.

The callback must return `CONFID_OK` on success, `CONFID_ERR` on error or `CONFID_DELAYED_RESPONSE` if the reply value is not yet available.

`find_next()`

This optional callback can be registered to optimize cases where ConfD wants to start a list traversal at some other point than at the first entry of the list, or otherwise make a "jump" in a list traversal. If the callback is not registered, ConfD will use a sequence of `get_next()` calls to find the desired list entry.

Where the `get_next()` callback provides a *next* parameter to indicate which keys should be returned, this callback instead provides a *type* parameter and a set of values to indicate which keys should be returned. Just like for `get_next()`, the callback should reply by calling `confd_data_reply_next_key()` with the keys for the requested list entry.

The *keys* parameter is a pointer to a *nkeys* elements long array of key values, or secondary index-leaf values (see below). The *type* can have one of two values:

`CONFID_FIND_NEXT`

The callback should always reply with the key values for the first list entry *after* the one indicated by the *keys* array, and a *next* value appropriate for retrieval of subsequent entries. The *keys* array may not correspond to an actual existing list entry - the callback must return the keys for the first existing entry that is "later" in the list order than the keys provided by the callback. Furthermore the number of values provided in the array (*nkeys*) may be fewer than the number of keys (or number of index-leaves for a secondary-index) in the data model, possibly even zero. This means that only the first *nkeys* values are provided, and the remaining ones should be taken to have a value "earlier" than the value for any existing list entry.

`CONFID_FIND_SAME_OR_NEXT`

If the values in the *keys* array completely identify an actual existing list entry, the callback should reply with the keys for this list entry and a corresponding *next* value. Otherwise the same logic as described for `CONFID_FIND_NEXT` should be used.

The `dp/find_next` example in the bundled examples collection has an implementation of the `find_next()` callback for a list with two integer keys. It shows how the *type* value and the

provided keys need to be combined in order to find the requested entry - or find that no entry matching the request exists.

If the list has a `tailf:secondary-index` statement (see [tailf_yang_extensions\(5\)](#)), the callback must examine the value of the `tctx->secondary_index` variable, as described for the `get_next()` callback. If `tctx->secondary_index` has a value greater than 0, the `keys` and `nkeys` parameters do not represent key values, but instead values for the index leafs specified by the `tailf:index-leafs` statement for the secondary index. The callback should however still reply with the actual key values for the list entry in the `confd_data_reply_next_key()` call.

Once we have called `confd_data_reply_next_key()`, ConfD will use `get_next()` (or `get_next_object()`) for any subsequent entry-by-entry list traversal - however we can request that this traversal should be done using `find_next()` (or `find_next_object()`) instead, by passing -1 for the `next` parameter to `confd_data_reply_next_key()`. In this case ConfD will always invoke `find_next()/find_next_object()` with `type` `CONFDFINDNEXT`, and the (complete) set of keys from the previous reply.



Note

In the case of list traversal by means of a secondary index, the secondary index values must be unique for entry-by-entry traversal with `find_next()/find_next_object()` to be possible. Thus we can not pass -1 for the `next` parameter to `confd_data_reply_next_key()` in this case if the secondary index values are not unique.

To signal that no entry matching the request exists, i.e. we have reached the end of the list while evaluating the request, we reply with a NULL pointer as the key value in the `confd_data_reply_next_key()` function.



Note

For a list that does not specify a non-default sort order by means of a `ordered-by` user or `tailf:sort-order` statement, ConfD assumes that list entries are ordered strictly by increasing key values.

If we have registered `find_next()` (or `find_next_object()`), it is not strictly necessary to also register `get_next()` (or `get_next_object()`) - except for the case of traversal by secondary index when the secondary index values are not unique, see above. If a northbound agent does a `get_next` request, and neither `get_next()` nor `get_next_object()` is registered, ConfD will instead invoke `find_next()` (or `find_next_object()`), the same way as if -1 had been passed for the `next` parameter to `confd_data_reply_next_key()` as described above - the actual `next` value passed is ignored. The very first `get_next` request for a traversal (i.e. where the `next` parameter would be -1) will cause a `find_next` invocation with `type` `CONFDFINDNEXT` and `nkeys == 0`, i.e. no keys provided.

The callback must return `CONFDFINDNEXT` on success, `CONFDFINDERR` on error or `CONFDFINDDELAYEDRESPONSE` if the reply value is not yet available. In the latter case the application must at a later stage call `confd_data_reply_next_key()`.

`num_instances()`

This callback can optionally be implemented. The purpose is to return the number of entries in a list. If the callback is set to NULL, whenever ConfD needs to calculate the number of entries in a certain list, ConfD will iterate through the entries by means of consecutive calls to the `get_next()` callback.

If we have a large number of entries *and* it is computationally cheap to calculate the number of entries in a list, it may be worth the effort to implement this callback for performance reasons.

The number of entries is returned in an `confd_value_t` value of type `C_INT32`. The value is returned through a call to `confd_data_reply_value()`, see code example below:

```
int num_instances;
confd_value_t v;

CONFID_SET_INT32(&v, num_instances);
confd_data_reply_value(trans_ctx, &v);
return CONFID_OK;
```

Must return `CONFID_OK` on success, `CONFID_ERR` on error or `CONFID_DELAYED_RESPONSE`.

`get_object()`

The implementation of this callback is also optional. The purpose of the callback is to return an entire object, i.e. a list entry, in one swoop. If the callback is not implemented, ConfD will retrieve the whole object through a series of calls to `get_elem()`.

The callback will only be called for list entries - i.e. `get_elem()` is still needed for leafs that are not defined in a list, but if there are no such leafs in the part of the data model covered by a given callpoint, the `get_elem()` callback may be omitted when `get_object()` is registered. This has the drawback that ConfD will have to invoke `get_object()` even if only a single leaf in a list entry is needed though, e.g. for the existence test mentioned for `get_elem()`.

When ConfD invokes the `get_elem()` callback, it is the responsibility of the application to issue calls to the reply function `confd_data_reply_value()`. The `get_object()` callback cannot use this function since it needs to return a sequence of values. The `get_object()` callback must use either the `confd_data_reply_value_array()` function or the `confd_data_reply_tag_value_array()` function. See the description of these functions below for the details of the arguments passed. If the entry requested does not exist, the callback must call `confd_data_reply_not_found()`.

Remember, the callback `exists_optional()` must always be implemented when we have presence containers or leafs of type `empty`. If we also choose to implement the `get_object()` callback, ConfD can sometimes derive the existence of such a node through a previous call to `get_object()`. This is however not always the case, thus even if we implement `get_object()`, we must also implement `exists_optional()` if we have such nodes.

If we pass an array of values which does not comply with the rules for the above functions, ConfD will notice and an error is reported to the agent which issued the request. A message is also logged to ConfD's `developerLog`.

The callback must return `CONFID_OK` on success, `CONFID_ERR` on error or `CONFID_DELAYED_RESPONSE` if the reply value is not yet available.

`get_next_object()`

The implementation of this callback is also optional. Similar to the `get_object()` callback the purpose of this callback is to return an entire object, or even multiple objects, in one swoop. It combines the functionality of `get_next()` and `get_object()` into a single callback, and adds the possibility to return multiple objects. Thus we need only implement this callback if it is very important to be able to traverse a list very fast. If the callback is not implemented, ConfD will retrieve the whole object through a series of calls to `get_next()` and consecutive calls to either `get_elem()` or `get_object()`.

When we have registered `get_next_object()`, it is not strictly necessary to also register `get_next()`, but omitting `get_next()` may have a serious performance impact, since there are cases (e.g. CLI tab completion) when ConfD only wants to retrieve the keys for a list. In such a case, if we have only registered `get_next_object()`, all the data for the list will be

retrieved, but everything except the keys will be discarded. Also note that even if we have registered `get_next_object()`, at least one of the `get_elem()` and `get_object()` callbacks must be registered.

Similar to the `get_next()` callback, if the `next` parameter is `-1` ConfD wants to retrieve the first entry in the list.

Similar to the `get_next()` callback, if the `tctx->secondary_index` parameter is greater than `0` ConfD wants to retrieve the entries in the order defined by the secondary index.

Similar to the `get_object()` callback, `get_next_object()` needs to reply with an entire object expressed as either an array of `confd_value_t` values or an array of `confd_tag_value_t` values. It must also indicate which is the `next` entry in the list similar to the `get_next()` callback. The two functions `confd_data_reply_next_object_array()` and `confd_data_reply_next_object_tag_value_array()` are used to convey the return values for one object from the `get_next_object()` callback.

If we want to reply with multiple objects, we must instead use one of the functions `confd_data_reply_next_object_arrays()` and `confd_data_reply_next_object_tag_value_arrays()`. These functions take an "array of object arrays", where each element in the array corresponds to the reply for a single object with `confd_data_reply_next_object_array()` and `confd_data_reply_next_object_tag_value_array()`, respectively.

If we pass an array of values which does not comply with the rules for the above functions, ConfD will notice and an error is reported to the agent which issued the request. A message is also logged to ConfD's `developerLog`.

The callback must return `CONF_OK` on success, `CONF_ERR` on error or `CONF_DELAYED_RESPONSE` if the reply value is not yet available.

`find_next_object()`

The implementation of this callback is also optional. It relates to `get_next_object()` in exactly the same way as `find_next()` relates to `get_next()`. I.e. instead of a parameter `next`, we get a `type` parameter and a set of key values, or secondary index-leaf values, to indicate which object or objects to return to ConfD via one of the reply functions.

Similar to the `get_next_object()` callback, if the `tctx->secondary_index` parameter is greater than `0` ConfD wants to retrieve the entries in the order defined by the secondary index. And as described for the `find_next()` callback, in this case the `keys` and `nkeys` parameters represent values for the index leafs specified by the `tailf:index-leafs` statement for the secondary index.

Similar to the `get_next_object()` callback, the callback can use any of the functions `confd_data_reply_next_object_array()`, `confd_data_reply_next_object_tag_value_array()`, `confd_data_reply_next_object_arrays()`, and `confd_data_reply_next_object_tag_value_arrays()` to return one or more objects to ConfD.

If we pass an array of values which does not comply with the rules for the above functions, ConfD will notice and an error is reported to the agent which issued the request. A message is also logged to ConfD's `developerLog`.

The callback must return `CONF_OK` on success, `CONF_ERR` on error or `CONF_DELAYED_RESPONSE` if the reply value is not yet available.

`get_case()`

This callback only needs to be implemented if we use the YANG `choice` statement in the part of the data model that our data provider is responsible for, but when we use choice, the callback is required. It should return the currently selected `case` for the choice given by the `choice` argument - `kp` is the path to the container or list entry where the choice is defined.

In the general case, where there may be multiple levels of `choice` statements without intervening `container` or `list` statements in the data model, the choice is represented as an array of `confd_value_t` elements with the type `C_XMLTAG`, terminated by an element with the type `C_NOEXISTS`. This array gives a reversed path with alternating choice and case names, from the data node given by `kp` to the specific choice that the callback request pertains to - similar to how a `confd_hkeypath_t` gives a path through the data tree.

If we don't have such "nested" choices in the data model, we can ignore this array aspect, and just treat the `choice` argument as a single `confd_value_t` value. The case is always represented as a `confd_value_t` with the type `C_XMLTAG`. I.e. we can use `CONFID_GET_XMLTAG()` to get the choice tag from `choice` and `CONFID_SET_XMLTAG()` to set the case tag for the reply value. The callback should use `confd_data_reply_value()` to return the case value to ConfD, or `confd_data_reply_not_found()` for an optional choice without default case if no case is currently selected. If an optional choice with default case does not have a selected case, the callback should use `confd_data_reply_value()` with a value of type `C_DEFAULT`.

Must return `CONFID_OK` on success, `CONFID_ERR` on error, or `CONFID_DELAYED_RESPONSE`.

`set_case()`

This callback is completely optional, and will only be invoked (if registered) if we use the YANG `choice` statement and provide configuration data. The callback sets the currently selected `case` for the choice given by the `kp` and `choice` arguments, and is mainly intended to make it easier to support the `get_case()` callback. ConfD will additionally invoke the `remove()` callback for all nodes in the previously selected case, i.e. if we register `set_case()`, we do not need to analyze `set_elem()` callbacks to determine the currently selected case, or figure out which nodes that should be deleted.

For a choice without a mandatory `true` statement, it is possible to have no case at all selected. To indicate that the previously selected case should be deleted without selecting another case, the callback will be invoked with `NULL` for the `caseval` argument.

The callback must return `CONFID_OK` on success, `CONFID_ERR` on error, `CONFID_DELAYED_RESPONSE` or `CONFID_ACCUMULATE`.

`get_attrs()`

This callback only needs to be implemented for callpoints specified for configuration data, and only if attributes are enabled in the ConfD configuration (`/confdConfig/enableAttributes` set to `true`). These are the currently supported attributes:

```
/* CONFID_ATTR_TAGS: value is C_LIST of C_BUF/C_STR */
#define CONFID_ATTR_TAGS      0x80000000
/* CONFID_ATTR_ANNOTATION: value is C_BUF/C_STR */
#define CONFID_ATTR_ANNOTATION 0x80000001
/* CONFID_ATTR_INACTIVE: value is C_BOOL 1 (i.e. "true") */
#define CONFID_ATTR_INACTIVE   0x00000000
```

The `attrs` parameter is an array of attributes of length `num_attrs`, giving the requested attributes - if `num_attrs` is 0, all attributes are requested. If the node given by `kp` does not exist, the callback should reply by calling `confd_data_reply_not_found()`, otherwise it should call `confd_data_reply_attrs()`, even if no attributes are set.



Note It is very important to observe this distinction, i.e. to use `confd_data_reply_not_found()` when the node doesn't exist, since ConfD may use `get_attrs()` as an existence check when attributes are enabled. (This avoids doing one callback request for existence check and another to collect the attributes.)

Must return `CONFD_OK` on success, `CONFD_ERR` on error, or `CONFD_DELAYED_RESPONSE`.

`set_attr()`

This callback also only needs to be implemented for callpoints specified for configuration data, and only if attributes are enabled in the ConfD configuration (`/confdConfig/enableAttributes` set to true). See `get_attrs()` above for the supported attributes.

The callback should set the attribute `attr` for the node given by `kp` to the value `v`. If the callback is invoked with `NULL` for the value argument, it means that the attribute should be deleted.

The callback must return `CONFD_OK` on success, `CONFD_ERR` on error, `CONFD_DELAYED_RESPONSE` or `CONFD_ACCUMULATE`.

`move_after()`

This callback only needs to be implemented if we provide configuration data that has YANG lists with a `ordered-by user` statement. The callback moves the list entry given by `kp`. If `prevkeys` is `NULL`, the entry is moved first in the list, otherwise it is moved after the entry given by `prevkeys`. In this case `prevkeys` is a pointer to an array of key values identifying an entry in the list. The array is terminated with an element that has type `C_NOEXISTS`.

The callback must return `CONFD_OK` on success, `CONFD_ERR` on error, `CONFD_DELAYED_RESPONSE` or `CONFD_ACCUMULATE`.

`write_all()`

This callback will only be invoked for a transaction hook specified with `tailf:invocation-mode per-transaction;` - see the chapter [Transformations, Hooks, Hidden Data and Symlinks](#) in the User Guide. It is also the only callback that is invoked for such a hook. The callback is expected to make all the modifications to the current transaction that hook functionality requires. The `kp` parameter is currently always `NULL`, since the callback does not pertain to any particular data node.

The callback must return `CONFD_OK` on success, `CONFD_ERR` on error, or `CONFD_DELAYED_RESPONSE`.

The six write callbacks (excluding `write_all()`), namely `set_elem()`, `create()`, `remove()`, `set_case()`, `set_attr()`, and `move_after()` may return the value `CONFD_ACCUMULATE`. If `CONFD_ACCUMULATE` is returned the library will accumulate the written values as a linked list of operations. This list can later be traversed in either of the transaction callbacks `prepare()` or `commit()`.

This provides trivial transaction support for applications that want to implement the ConfD two-phase commit protocol but lacks an underlying database with proper transaction support. The write operations are available as a linked list of `confd_tr_item` structs:

```
struct confd_tr_item {
    char *callpoint;
    enum confd_tr_op op;
    confd_hkeypath_t *hkp;
    confd_value_t *val;
    confd_value_t *choice; /* only for set_case */
}
```



```

    uint32_t attr;          /* only for set_attr */
    struct confd_tr_item *next;
};

```

The list is available in the transaction context in the field `accumulated`. The entire list and its content will be automatically freed by the library once the transaction finishes.

```

int confd_register_range_data_cb(struct confd_daemon_ctx *dx, const
struct confd_data_cbs *data, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);

```

This is a variant of `confd_register_data_cb()` which registers a set of callbacks for a range of list entries. There can thus be multiple sets of C functions registered on the same callpoint, even by different daemons. The `lower` and `upper` parameters are two `numkeys` long arrays of key values, which define the endpoints of the list range. It is also possible to do a "default" registration, by giving `lower` and `upper` as NULL (`numkeys` is ignored). The callbacks for the default registration will be invoked when the keys are not in any of the explicitly registered ranges.

The `fmt` and remaining parameters specify a string path for the list that the keys apply to, in the same form as for the [confd_lib_maapi\(3\)](#) and [confd_lib_cdb\(3\)](#) functions. However if the list is a sublist to another list, the key element for the parent list(s) may be completely omitted, to indicate that the registration applies to all entries for the parent list(s) (similar to CDB subscription paths).

An example that registers one set of callbacks for the range `/servers/server{aaa} - /servers/server{mzz}` and another set for `/servers/server{naa} - /servers/server{zzz}`:

```

confd_value_t lower, upper;

CONFD_SET_STR(&lower, "aaa");
CONFD_SET_STR(&upper, "mzz");
if (confd_register_range_data_cb(dctx, &data_cb1, &lower, &upper, 1,
                                "/servers/server") == CONFD_ERR)
    confd_fatal("Failed to register data cb\n");

CONFD_SET_STR(&lower, "naa");
CONFD_SET_STR(&upper, "zzz");
if (confd_register_range_data_cb(dctx, &data_cb2, &lower, &upper, 1,
                                "/servers/server") == CONFD_ERR)
    confd_fatal("Failed to register data cb\n");

```

In this example, as in most cases where this function is used, the data model defines a list with a single key, and `numkeys` is thus always 1. However it can also be used for lists that have multiple keys, in which case the `upper` and `lower` arrays may be populated with multiple keys, upto however many keys the data model specifies for the list, and `numkeys` gives the number of keys in the arrays. If fewer keys than specified in the data model are given, the registration covers all possible values for the remaining keys, i.e. they are effectively wildcarded.

While traversal of a list with range registrations will always invoke e.g. `get_next()` only for actually registered ranges, it is also possible that a request from a northbound interface is made for data in a specific list entry. If the registrations do not cover all possible key values, such a request could be for a list entry that does not fall in any of the registered ranges, which will result in a "no registration" error. To avoid the error, we can either restrict the type of the keys such that only values that fall in the registered ranges are valid, or, for operational data, use a "default" registration as described above. In this case the daemon with the "default" registration would just reply with `confd_data_reply_not_found()` for all requests for specific data, and `confd_data_reply_next_key()` with NULL for the key values for all `get_next()` etc requests.

**Note**

For a given callpoint name, there can only be either one non-range registration or a number of range registrations that all pertain to the same list. If a range registration is done after a non-range registration or vice versa, or if a range registration is done with a different list path than earlier range registrations, the latest registration completely replaces the earlier one(s). If we want to register for the same ranges in different lists, we must thus have a unique callpoint for each list.

**Note**

Range registrations can not be used for lists that have the `tailf:secondary-index` extension, since there is no way for ConfD to traverse the registrations in secondary-index order.

```
int confd_register_usess_cb(struct confd_daemon_ctx *dx, const struct
confd_usess_cbs *ucb);
```

This function can be used to register information callbacks that are invoked for user session start and stop. The struct `confd_usess_cbs` is defined as:

```
struct confd_usess_cbs {
    void (*start)(struct confd_daemon_ctx *dx,
                  struct confd_user_info *uinfo);
    void (*stop)(struct confd_daemon_ctx *dx,
                 struct confd_user_info *uinfo);
};
```

Both callbacks are optional. They can be used e.g. for a multi-threaded daemon to manage a pool of worker threads, by allocating worker threads to user sessions. In this case we would ideally allocate a worker thread the first time an `init()` callback for a given user session requires a worker socket to be assigned, and use only the `stop()` usess callback to release the worker thread - using the `start()` callback to allocate a worker thread would often mean that we allocated a thread that was never used. The `u_opaque` element in the struct `confd_user_info` can be used to manage such allocations.

**Note**

These callbacks will only be invoked if the daemon has also registered other callbacks. Furthermore, as an optimization, ConfD will delay the invocation of the `start()` callback until some other callback is invoked. This means that if no other callbacks for the daemon are invoked for the duration of a user session, neither `start()` nor `stop()` will be invoked for that user session. If we want timely notification of start and stop for all user sessions, we can subscribe to `CONFID_NOTIF_AUDIT` events, see [confd_lib_events\(3\)](#).

**Note**

When we call `confd_register_done()` (see below), the `start()` callback (if registered) will be invoked for each user session that already exists.

```
int confd_register_done(struct confd_daemon_ctx *dx);
```

When we have registered all the callbacks for a daemon (including the other types described below if we have them), we must call this function to synchronize with ConfD. No callbacks will be invoked until it has been called, and after the call, no further registrations are allowed.

```
int confd_fd_ready(struct confd_daemon_ctx *dx, int fd);
```

The database application owns all data provider sockets to ConfD and is responsible for the polling of these sockets. When one of the ConfD sockets has I/O ready to read, the application must invoke `confd_fd_ready()` on the socket. This function will:

- Read data from ConfD
- Unmarshal this data
- Invoke the right callback with the right arguments

When this function reads the request from ConfD it will block on `read()`, thus if it is important for the application to have nonblocking I/O, the application must dispatch I/O from ConfD in a separate thread.

The function returns the return value from the callback function, normally `CONFDFD_OK` (0), or `CONFDFD_ERR` (-1) on error and `CONFDFD_EOF` (-2) when the socket to ConfD has been closed. Thus `CONFDFD_ERR` can mean either that the callback function that was invoked returned `CONFDFD_ERR`, or that some error condition occurred within the `confd_fd_ready()` function. These cases can be distinguished via `confd_errno`, which will be set to `CONFDFD_ERR_EXTERNAL` if `CONFDFD_ERR` comes from the callback function. Thus a correct call to `confd_fd_ready()` looks like:

```
struct pollfd set[n];
/* ..... */

if (set[0].revents & POLLIN) {
    if ((ret = confd_fd_ready(dctx, mysock)) == CONFDFD_EOF) {
        confd_fatal("ConfD socket closed\n");
    } else if (ret == CONFDFD_ERR &&
               confd_errno != CONFDFD_ERR_EXTERNAL) {
        confd_fatal("Error on ConfD socket request: %s (%d): %s\n",
                    confd_strerror(confd_errno), confd_errno,
                    confd_lasterr());
    }
}
```

Errors: `CONFDFD_ERR_MALLOC`, `CONFDFD_ERR_OS`, `CONFDFD_ERR_PROTOUSAGE`, `CONFDFD_ERR_EXTERNAL`

```
void confd_trans_set_fd(struct confd_trans_ctx *tctx, int sock);
```

Associate a worker socket with the transaction, or validation phase. This function must be called in the transaction and validation `init()` callbacks - a minimal implementation of a transaction `init()` callback looks like:

```
static int init(struct confd_trans_ctx *tctx)
{
    confd_trans_set_fd(tctx, workersock);
    return CONFDFD_OK;
}
```

```
int confd_data_reply_value(struct confd_trans_ctx *tctx, const
confd_value_t *v);
```

This function is used to return a single data item to ConfD.

Errors: `CONFDFD_ERR_PROTOUSAGE`, `CONFDFD_ERR_MALLOC`, `CONFDFD_ERR_OS`, `CONFDFD_ERR_BADTYPE`

```
int confd_data_reply_value_array(struct confd_trans_ctx *tctx, const
confd_value_t *vs, int n);
```

This function is used to return an array of values, corresponding to a complete list entry, to ConfD. It can be used by the optional `get_object()` callback. The `vs` array is populated with n values according to the specification of the Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

In the easiest case, similar to the "servers" example above, we can construct a reply array as follows:

```
struct in_addr ip4 = my_get_ip(...);
confd_value_t ret[3];

CONFD_SET_STR(&ret[0], "www");
CONFD_SET_IPV4(&ret[1], ip4);
CONFD_SET_UINT16(&ret[2], 80);
confd_data_reply_value_array(tctx, ret, 3);
```

Any containers inside the object must also be passed in the array. For example an entry in the `b` list used in the explanation for `exists_optional()` would have to be passed as:

```
confd_value_t ret[4];

CONFD_SET_STR(&ret[0], "b_name");
CONFD_SET_XMLTAG(&ret[1], myprefix_opt, myprefix_ns);
CONFD_SET_INT32(&ret[2], 77);
CONFD_SET_NOEXISTS(&ret[3]);

confd_data_reply_value_array(tctx, ret, 4);
```

Thus, a container or a leaf of type empty must be passed as its equivalent XML tag if it exists. If a presence container or leaf of type empty does not exist, it must be passed as a value of `C_NOEXISTS`. In the example above, the leaf `foo` does not exist, thus the contents of position 3 in the array.

If a presence container does not exist, its non existing values must not be passed - it suffices to say that the container itself does not exist. In the example above, the `opt` container did exist and thus we also had to pass the contained value(s), the `ii` leaf.

Hence, the above example represents:

```
<b>
  <name>b_name</name>
  <opt>
    <ii>77</ii>
  </opt>
</b>
```

```
int confd_data_reply_tag_value_array(struct confd_trans_ctx *tctx,
const confd_tag_value_t *tvs, int n);
```

This function is used to return an array of values, corresponding to a complete list entry, to ConfD. It can be used by the optional `get_object()` callback. The `tvs` array is populated with n values according to the Tagged Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

I.e. the difference from `confd_data_reply_value_array()` is that the values are tagged with the node names from the data model - this means that non-existing values can simply be omitted from the array, per the specification above. Additionally the key leafs can be omitted, since they are already known by ConfD - if the key leafs are included, they will be ignored. Finally, in e.g. the case of a container with both config and non-config data, where the config data is in CDB and only the non-config data provided by the callback, the config elements can be omitted (for `confd_data_reply_value_array()` they must be included as `C_NOEXISTS` elements).

However, although the tagged value array format can represent nested lists, these must not be passed via this function, since the `get_object()` callback only pertains to a single entry of one list. Nodes representing sub-lists must thus be omitted from the array, and ConfD will issue separate `get_object()` invocations to retrieve the data for those.

Using the same examples as above, in the "servers" case, we can construct a reply array as follows:

```
struct in_addr ip4 = my_get_ip(.....);
confd_tag_value_t ret[2];
int n = 0;

CONFID_SET_TAG_IPV4(&ret[n], myprefix_ip, ip4); n++;
CONFID_SET_TAG_UINT16(&ret[n], myprefix_port, 80); n++;
confd_data_reply_tag_value_array(tctx, ret, n);
```

An entry in the `b` list used in the explanation for `exists_optional()` would be passed as:

```
confd_tag_value_t ret[3];
int n = 0;

CONFID_SET_TAG_XMLBEGIN(&ret[n], myprefix_opt, myprefix_ns); n++;
CONFID_SET_TAG_INT32(&ret[n], myprefix_ii, 77); n++;
CONFID_SET_TAG_XMLEND(&ret[n], myprefix_opt, myprefix_ns); n++;
confd_data_reply_tag_value_array(tctx, ret, n);
```

The `C_XMLEND` element is not strictly necessary in this case, since there are no subsequent elements in the array. However it would have been required if the optional `foo` leaf had existed, thus it is good practice to always include both the `C_XMLBEGIN` and `C_XMLEND` elements for nested containers (if they exist, that is - otherwise neither must be included).

```
int confd_data_reply_next_key(struct confd_trans_ctx *tctx, const
confd_value_t *v, int num_vals_in_key, long next);
```

This function is used by the `get_next()` and `find_next()` callbacks to return the next key. A list may have multiple key leafs specified in the data model. The parameter `num_vals_in_key` indicates the number of key values, i.e. the length of the `v` array. In the typical case with all lists having just a single key leaf specified, `num_vals_in_key` is always 1.

The `long next` will be passed into the next invocation of the `get_next()` callback if it has a value other than -1. Thus this value provides a means for the application to traverse the data. Since this is long it is possible to pass a `void*` pointing to the next list entry in the application - effectively passing a pointer to `confd` and getting it back in the next invocation of `get_next()`.

To indicate that no more entries exist, we reply with a `NULL` pointer for the `v` array. The values of the `num_vals_in_key` and `next` parameters are ignored in this case.

Passing the value -1 for `next` has a special meaning. It tells ConfD that we want the next request for this list traversal to use the `find_next()` (or `find_next_object()`) callback instead of `get_next()` (or `get_next_object()`).



Note

In the case of list traversal by means of a secondary index, the secondary index values must be unique for entry-by-entry traversal with `find_next()/find_next_object()` to be possible. Thus we can not pass -1 for the `next` parameter in this case if the secondary index values are not unique.

Errors: `CONFID_ERR_PROTOUSAGE`, `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_BADTYPE`

```
int confd_data_reply_not_found(struct confd_trans_ctx *tctx);
```

This function is used by the `get_elem()` and `exists_optional()` callbacks to indicate to ConfD that a list entry or node does not exist.

Errors: CONFID_ERR_PROTOUSAGE, CONFID_ERR_MALLOC, CONFID_ERR_OS

```
int confd_data_reply_found(struct confd_trans_ctx *tctx);
```

This function is used by the `exists_optional()` callback to indicate to ConfD that a node does exist.

Errors: CONFID_ERR_PROTOUSAGE, CONFID_ERR_MALLOC, CONFID_ERR_OS

```
int confd_data_reply_next_object_array(struct confd_trans_ctx *tctx,  
const confd_value_t *v, int n, long next);
```

This function is used by the optional `get_next_object()` and `find_next_object()` callbacks to return an entire object including its keys, as well as the `next` parameter that has the same function as for `confd_data_reply_next_key()`. It combines the functions of `confd_data_reply_next_key()` and `confd_data_reply_value_array()`.

The array of `confd_value_t` elements must be populated in exactly the same manner as for `confd_data_reply_value_array()` and the `long next` is used in the same manner as the equivalent `next` parameter in `confd_data_reply_next_key()`. To indicate the end of the list we - similar to `confd_data_reply_next_key()` - pass a NULL pointer for the value array.

If we are replying to a `get_next_object()` or `find_next_object()` request for an operational data list without keys (see the [Operational Data chapter in the User Guide](#)), we must include the "pseudo" key in the array, as the first element (i.e. preceding the actual leafs from the data model).

Errors: CONFID_ERR_PROTOUSAGE, CONFID_ERR_MALLOC, CONFID_ERR_OS,
CONFID_ERR_BADTYPE

```
int confd_data_reply_next_object_tag_value_array(struct confd_trans_ctx  
*tctx, const confd_tag_value_t *tv, int n, long next);
```

This function is used by the optional `get_next_object()` and `find_next_object()` callbacks to return an entire object including its keys, as well as the `next` parameter that has the same function as for `confd_data_reply_next_key()`. It combines the functions of `confd_data_reply_next_key()` and `confd_data_reply_tag_value_array()`.

Similar to how the `confd_data_reply_value_array()` has its companion function `confd_data_reply_tag_value_array()` if we want to return an object as an array of `confd_tag_value_t` values instead of an array of `confd_value_t` values, we can use this function instead of `confd_data_reply_next_object_array()` when we wish to return values from the `get_next_object()` callback.

The array of `confd_tag_value_t` elements must be populated in exactly the same manner as for `confd_data_reply_tag_value_array()` (except that the key values must be included), and the `long next` is used in the same manner as the equivalent `next` parameter in `confd_data_reply_next_key()`. The key leafs must always be given as the first elements of the array, and in the order specified in the data model. To indicate the end of the list we - similar to `confd_data_reply_next_key()` - pass a NULL pointer for the value array.

If we are replying to a `get_next_object()` or `find_next_object()` request for an operational data list without keys (see the [Operational Data chapter in the User Guide](#)), the "pseudo" key must be included, as the first element in the array, with a tag value of 0 - i.e. it can be set with code like this:

```
confd_tag_value_t tv[7];
```

```
CONFID_SET_TAG_INT64(&tv[0], 0, 42);
```

Errors: CONFID_ERR_PROTOUSAGE, CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADTYPE

```
int confd_data_reply_next_object_arrays(struct confd_trans_ctx *tctx,
const struct confd_next_object *obj, int nobj, int timeout_millisecs);
```

This function is used by the optional `get_next_object()` and `find_next_object()` callbacks to return multiple objects including their keys, in `confd_value_t` form. The struct `confd_next_object` is defined as:

```
struct confd_next_object {
    confd_value_t *v;
    int n;
    long next;
};
```

I.e. it corresponds exactly to the data provided for a call of `confd_data_reply_next_object_array()`. The parameter `obj` is a pointer to an `nobj` elements long array of such structs. We can also pass a timeout value for ConfD's caching of the returned data via `timeout_millisecs`. If we pass 0 for this parameter, the value configured via `/confdConfig/capi/objectCacheTimeout` in `confd.conf` (see [confd.conf\(5\)](#)) will be used.

The cache in ConfD may become invalid (e.g. due to timeout) before all the returned list entries have been used, and ConfD may then need to issue a new callback request based on an "intermediate" next value. This is done exactly as for the single-entry case, i.e. if `next` is `-1`, `find_next_object()` (or `find_next()`) will be used, with the keys from the "previous" entry, otherwise `get_next_object()` (or `get_next()`) will be used, with the given next value.

Thus a data provider can choose to give next values that uniquely identify list entries if that is convenient, or otherwise use `-1` for all next elements - or a combination, e.g. `-1` for all but the last entry. If any next value is given as `-1`, at least one of the `find_next()` and `find_next_object()` callbacks must be registered.

To indicate the end of the list we can either pass a NULL pointer for the `obj` array, or pass an array where the last struct `confd_next_object` element has the `v` element set to NULL. The latter is preferable, since we can then combine the final list entries with the end-of-list indication in the reply to a single callback invocation.



Note

When next values other than `-1` are used, these must remain valid even after the end of the list has been reached, since ConfD may still need to issue a new callback request based on an "intermediate" next value as described above. They can be discarded (e.g. allocated memory released) when a new `get_next_object()` or `find_next_object()` callback request for the same list in the same transaction has been received, or at the end of the transaction.



Note

In the case of list traversal by means of a secondary index, the secondary index values must be unique for entry-by-entry traversal with `find_next_object()/find_next()` to be possible. Thus we can not use `-1` for the next element in this case if the secondary index values are not unique.

Errors: CONFID_ERR_PROTOUSAGE, CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_BADTYPE

```
int confd_data_reply_next_object_tag_value_arrays(struct
confd_trans_ctx *tctx, const struct confd_tag_next_object *tobj, int
nobj, int timeout_millisecs);
```

This function is used by the optional `get_next_object()` and `find_next_object()` callbacks to return multiple objects including their keys, in `confd_tag_value_t` form. The struct `confd_tag_next_object` is defined as:

```
struct confd_tag_next_object {
    confd_tag_value_t *tv;
    int n;
    long next;
};
```

I.e. it corresponds exactly to the data provided for a call of `confd_data_reply_next_object_tag_value_array()`. The parameter `tobj` is a pointer to an `nobj` elements long array of such structs. We can also pass a timeout value for ConfD's caching of the returned data via `timeout_millisecs`. If we pass 0 for this parameter, the value configured via `/confdConfig/capi/objectCacheTimeout` in `confd.conf` (see [confd.conf\(5\)](#)) will be used.

The cache in ConfD may become invalid (e.g. due to timeout) before all the returned list entries have been used, and ConfD may then need to issue a new callback request based on an "intermediate" next value. This is done exactly as for the single-entry case, i.e. if `next` is `-1`, `find_next_object()` (or `find_next()`) will be used, with the keys from the "previous" entry, otherwise `get_next_object()` (or `get_next()`) will be used, with the given `next` value.

Thus a data provider can choose to give `next` values that uniquely identify list entries if that is convenient, or otherwise use `-1` for all `next` elements - or a combination, e.g. `-1` for all but the last entry. If any `next` value is given as `-1`, at least one of the `find_next()` and `find_next_object()` callbacks must be registered.

To indicate the end of the list we can either pass a NULL pointer for the `tobj` array, or pass an array where the last struct `confd_tag_next_object` element has the `tv` element set to NULL. The latter is preferable, since we can then combine the final list entries with the end-of-list indication in the reply to a single callback invocation.



Note

When `next` values other than `-1` are used, these must remain valid even after the end of the list has been reached, since ConfD may still need to issue a new callback request based on an "intermediate" `next` value as described above. They can be discarded (e.g. allocated memory released) when a new `get_next_object()` or `find_next_object()` callback request for the same list in the same transaction has been received, or at the end of the transaction.



Note

In the case of list traversal by means of a secondary index, the secondary index values must be unique for entry-by-entry traversal with `find_next_object()/find_next()` to be possible. Thus we can not use `-1` for the `next` element in this case if the secondary index values are not unique.

Errors: `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`

```
int confd_data_reply_attrs(struct confd_trans_ctx *tctx, const
confd_attr_value_t *attrs, int num_attrs);
```


This function is used by the `get_attrs()` callback to return the requested attribute values. The *attrs* array should be populated with *num_attrs* elements of type `confd_attr_value_t`, which is defined as:

```
typedef struct confd_attr_value {
    u_int32_t attr;
    confd_value_t v;
} confd_attr_value_t;
```

If multiple attributes were requested in the callback invocation, they should be given in the same order in the reply as in the request. Requested attributes that are not set should be omitted from the array. If none of the requested attributes are set, or no attributes at all are set when all attributes are requested, *num_attrs* should be given as 0, and the value of *attrs* is ignored.

Errors: `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`

```
int confd_delayed_reply_ok(struct confd_trans_ctx *tctx);
```

This function must be used to return the equivalent of `CONFD_OK` when the actual callback returned `CONFD_DELAYED_RESPONSE`. I.e. it is appropriate for a transaction callback, a data callback for a write operation, or a validation callback, when the result is successful.

Errors: `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int confd_delayed_reply_error(struct confd_trans_ctx *tctx, const char
*errstr);
```

This function must be used to return an error when the actual callback returned `CONFD_DELAYED_RESPONSE`. There are two cases where the value of *errstr* has a special significance:

"locked" after invocation of <code>trans_lock()</code>	This is equivalent to returning <code>CONFD_ALREADY_LOCKED</code> from the callback.
"in_use" after invocation of <code>write_start()</code> or <code>prepare()</code>	This is equivalent to returning <code>CONFD_IN_USE</code> from the callback.

In all other cases, calling `confd_delayed_reply_error()` is equivalent to calling `confd_trans_seterr()` with the *errstr* value and returning `CONFD_ERR` from the callback. It is also possible to first call `confd_trans_seterr()` (for the varargs format) or `confd_trans_seterr_extended()` etc (for [EXTENDED ERROR REPORTING](#) as described in [confd_lib_lib\(3\)](#)), and then call `confd_delayed_reply_error()` with `NULL` for *errstr*.

Errors: `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int confd_data_set_timeout(struct confd_trans_ctx *tctx, int
timeout_secs);
```

A data callback should normally complete "quickly", since e.g. the execution of a 'show' command in the CLI may require many data callback invocations. Thus it should be possible to set the `/confdConfig/capi/queryTimeout` in `confd.conf` (see above) such that it covers the longest possible execution time for any data callback. In some rare cases it may still be necessary for a data callback to have a longer execution time, and then this function can be used to extend (or shorten) the timeout for the current callback invocation. The timeout is given in seconds from the point in time when the function is called.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
void confd_trans_seterr(struct confd_trans_ctx *tctx, const char
*fmt, ...);
```

This function is used by the application to set an error string. The next transaction or data callback which returns CONFD_ERR will have this error description attached to it. This error may propagate to the CLI, the NETCONF manager, the Web UI or the log files depending on the situation. We also use this function to propagate warning messages from the `validate()` callback if we are doing semantic validation in C. The *fmt* argument is a printf style format string.

```
void confd_trans_seterr_extended(struct confd_trans_ctx *tctx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);
```

This function can be used to provide more structured error information from a transaction or data callback, see the section [EXTENDED ERROR REPORTING](#) in [confd_lib_lib\(3\)](#).

```
int confd_trans_seterr_extended_info(struct confd_trans_ctx *tctx,
enum confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);
```

This function can be used to provide structured error information in the same way as `confd_trans_seterr_extended()`, and additionally provide contents for the NETCONF <error-info> element. See the section [EXTENDED ERROR REPORTING](#) in [confd_lib_lib\(3\)](#).

```
void confd_db_seterr(struct confd_db_ctx *dbx, const char *fmt, ...);
```

This function is used by the application to set an error string. The next db callback function which returns CONFD_ERR will have this error description attached to it. This error may propagate to the CLI, the NETCONF manager, the Web UI or the log files depending on the situation. The *fmt* argument is a printf style format string.

```
void confd_db_seterr_extended(struct confd_db_ctx *dbx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);
```

This function can be used to provide more structured error information from a db callback, see the section [EXTENDED ERROR REPORTING](#) in [confd_lib_lib\(3\)](#).

```
int confd_db_seterr_extended_info(struct confd_db_ctx *dbx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);
```

This function can be used to provide structured error information in the same way as `confd_db_seterr_extended()`, and additionally provide contents for the NETCONF <error-info> element. See the section [EXTENDED ERROR REPORTING](#) in [confd_lib_lib\(3\)](#).

```
int confd_db_set_timeout(struct confd_db_ctx *dbx, int timeout_secs);
```

Some of the DB callbacks registered via `confd_register_db_cb()`, e.g. `copy_running_to_startup()`, may require a longer execution time than others, and in these cases the timeout specified for `/confdConfig/capi/newSessionTimeout` may be insufficient. This

function can then be used to extend the timeout for the current callback invocation. The timeout is given in seconds from the point in time when the function is called.

```
int confd_aaa_reload(const struct confd_trans_ctx *tctx);
```

When the ConfD AAA tree is populated by an external data provider (see the [AAA chapter in the User Guide](#)), this function can be used by the data provider to notify ConfD when there is a change to the AAA data. I.e. it is an alternative to executing the command **confd --clear-aaa-cache**. See also `maapi_aaa_reload()` in [confd_lib_maapi\(3\)](#).

```
int confd_install_crypto_keys(struct confd_daemon_ctx* dtx);
```

It is possible to define DES3 and AES keys inside `confd.conf`. These keys are used by ConfD to encrypt data which is entered into the system which has either of the two builtin types `tailf:des3-cbc-encrypted-string` or `tailf:aes-cfb-128-encrypted-string`. See [confd_types\(3\)](#).

This function will copy those keys from ConfD (which reads `confd.conf`) into memory in the library. The parameter `dtx` is a daemon context which is connected through a call to `confd_connect()`.



Note

The function must be called before `confd_register_done()` is called. If this is impractical, or if the application doesn't otherwise use a daemon context, the equivalent function `maapi_install_crypto_keys()` may be more convenient to use, see [confd_lib_maapi\(3\)](#).

NCS SERVICE CALLBACKS

NCS service callbacks are invoked in a manner similar to the data callbacks described above, but require a registration for a service point, specified as `ncs:servicepoint` in the data model. The `init()` transaction callback must also be registered, and must use the `confd_trans_set_fd()` function to assign a worker socket for the transaction.

```
int ncs_register_service_cb(struct confd_daemon_ctx *dx, const struct ncs_service_cbs *scb);
```

This function registers the service callbacks. The struct `ncs_service_cbs` is defined as:

```
struct ncs_name_value {
    char *name;
    char *value;
};

enum ncs_service_operation {
    NCS_SERVICE_CREATE = 0,
    NCS_SERVICE_UPDATE = 1,
    NCS_SERVICE_DELETE = 2
};

struct ncs_service_cbs {
    char servicepoint[MAX_CALLPOINT_LEN];

    int (*pre_modification)(struct confd_trans_ctx *tctx,
                           enum ncs_service_operation op,
                           confd_hkeypath_t *kp,
                           struct ncs_name_value *proplist,
                           int num_props);
    int (*pre_lock_create)(struct confd_trans_ctx *tctx,
                           confd_hkeypath_t *kp,
                           struct ncs_name_value *proplist,
```

```

        int num_props, int fastmap_thandle);
int (*create)(struct confd_trans_ctx *tctx, confd_hkeypath_t *kp,
              struct ncs_name_value *proplist, int num_props,
              int fastmap_thandle);
int (*post_modification)(struct confd_trans_ctx *tctx,
                        enum ncs_service_operation op,
                        confd_hkeypath_t *kp,
                        struct ncs_name_value *proplist,
                        int num_props);
void *cb_opaque; /* private user data */
};

```

The `create()` callback is invoked inside NCS FASTMAP when creation or update of a service instance is committed. It should attach to the FASTMAP transaction by means of `maapi_attach2()` (see [confd_lib_maapi\(3\)](#)), passing the `fastmap_thandle` transaction handle as the `thandle` parameter to `maapi_attach2()`. The `usid` parameter for `maapi_attach2()` should be given as 0. To modify data in the FASTMAP transaction, the NCS-specific `maapi_shared_xxx()` functions must be used, see the section [NCS SPECIFIC FUNCTIONS](#) in the [confd_lib_maapi\(3\)](#) manual page.

The `pre_lock_create()` callback is invoked in the same way as the `create()` callback. The difference is that this callback is invoked outside the transaction lock of the current transaction, and may thus run in parallel with `pre_lock_create()` invocations in other transactions.



Note

A service can only register one of the two functions `create()` and `pre_lock_create()`

The `pre_modification()` and `post_modification()` callbacks are optional, and are invoked outside FASTMAP. `pre_modification()` is invoked before create, update, or delete of the service, as indicated by the `enum ncs_service_operation op` parameter. Conversely `post_modification()` is invoked after create, update, or delete of the service. These functions can be useful e.g. for allocations that should be stored and existing also when the service instance is removed.

All the callbacks receive a property list via the `proplist` and `num_props` parameters. This list is initially empty (`proplist == NULL` and `num_props == 0`), but it can be used to store and later modify persistent data outside the service model that might be needed.



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

```

int ncs_service_reply_proplist(struct confd_trans_ctx *tctx, const
struct ncs_name_value *proplist, int num_props);

```

This function must be called with the new property list, immediately prior to returning from the callback, if the stored property list should be updated. If a callback returns without calling `ncs_service_reply_proplist()`, the previous property list is retained. To completely delete the property list, call this function with the `num_props` parameter given as 0.

VALIDATION CALLBACKS

This library also supports the registration of callback functions on validation points in the data model. A validation point is a point in the data model where ConfD will invoke an external function to validate the associated data. The validation occurs before a transaction is committed. Similar to the state machine described for "external data bases" above where we install callback functions in the struct `confd_trans_cbs`,

we have to install callback functions for each validation point. It does not matter if the database is CDB or an external database, the validation callbacks described here work equally well for both cases.

```
void confd_register_trans_validate_cb(struct confd_daemon_ctx *dx,
const struct confd_trans_validate_cbs *vcbs);
```

This function installs two callback functions for the struct `confd_daemon_ctx`. One function that gets called when the validation phase starts in a transaction and one when the validation phase stops in a transaction. In the `init()` callback we can use the MAAPI api to attach to the running transaction, this way we can later on, freely traverse the configuration and read data. The data we will be reading through MAAPI (see [confd_lib_maapi\(3\)](#)) will be read from the shadow storage containing the *not-yet-committed* data.

The struct `confd_trans_validate_cbs` is defined as:

```
struct confd_trans_validate_cbs {
    int (*init)(struct confd_trans_ctx *tctx);
    int (*stop)(struct confd_trans_ctx *tctx);
};
```

It must thus be populated with two function pointers when we call this function.

The `init()` callback is conceptually invoked at the start of the validation phase, but just as for transaction callbacks, ConfD will as far as possible delay the actual invocation of the validation `init()` callback for a given daemon until it is required. This means that if none of the daemon's `validate()` callbacks need to be invoked (see below), `init()` and `stop()` will not be invoked either.

If we need to allocate memory or other resources for the validation this can also be done in the `init()` callback, with the resources being freed in the `stop()` callback. We can use the `t_opaque` element in the struct `confd_trans_ctx` to manage this, but in a daemon that implements both data and validation callbacks it is better to use the `v_opaque` element for validation, to be able to manage the allocations independently.

Similar to the `init()` callback for external data bases, we must in the `init()` callback associate a file descriptor with the transaction. This file descriptor will be used for the actual validation. Thus in a multi threaded application, we can have one thread performing validation for a transaction in parallel with other threads executing e.g. data callbacks. Thus a typical implementation of an `init()` callback for validation looks as:

```
static int init_validation(struct confd_trans_ctx *tctx)
{
    maapi_attach(maapi_socket, mtest_ns, tctx);
    confd_trans_set_fd(tctx, workersock);
    return CONFD_OK;
}
```

```
int confd_register_valpoint_cb(struct confd_daemon_ctx *dx, const
struct confd_valpoint_cb *vcb);
```

We must also install an actual validation function for each validation point, i.e. for each `tailf:validate` statement in the YANG data model.

A validation point has a name and an associated function pointer. The struct which must be populated for each validation point looks like:

```
struct confd_valpoint_cb {
    char valpoint[MAX_CALLPOINT_LEN];
    int (*validate)(struct confd_trans_ctx *tctx,
```

```

        confd_hkeypath_t *kp,
        confd_value_t *newval);
    void *cb_opaque;      /* private user data */
};

```



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

See the user guide chapter "Semantic validation" for code examples. The `validate()` callback can return `CONFD_OK` if all is well, or `CONFD_ERROR` if the validation fails. If we wish a message to accompany the error we must prior to returning from the callback, call `confd_trans_seterr()` or `confd_trans_seterr_extended()`.

The `cb_opaque` element can be used to pass arbitrary data to the callback, e.g. when the same callback is used for multiple validation points. It is made available to the callback via the element `vcb_opaque` in the transaction context (`tctx` argument), see the structure definition above.

If the `tailf:opaque` substatement has been used with the `tailf:validate` statement in the data model, the argument string is made available to the callback via the `validate_opaque` element in the transaction context.

We also have yet another special return value which can be used (only) from the `validate()` callback which is `CONFD_VALIDATION_WARN`. Prior to return of this value we must call `confd_trans_seterr()` which provides a string describing the warning. The warnings will get propagated to the transaction engine, and depending on where the transaction originates, ConfD may or may not act on the warnings. If the transaction originates from the CLI or the Web UI, ConfD will interactively present the user with a choice - whereby the transaction can be aborted.

If the transaction originates from NETCONF - which does not have any interactive capabilities - the warnings are ignored. The warnings are primarily intended to alert inexperienced users that attempt to make - dangerous - configuration changes. There can be multiple warnings from multiple validation points in the same transaction.

It is also possible to let the `validate()` callback return `CONFD_DELAYED_RESPONSE` in which case the application at a later stage must invoke either `confd_delayed_reply_ok()`, `confd_delayed_reply_error()` or `confd_delayed_reply_validation_warn()`.

In some cases it may be necessary for the validation callbacks to verify the availability of resources that will be needed if the new configuration is committed. To support this kind of verification, the `validation_info` element in the struct `confd_trans_ctx` can carry one of these flags:

CONFD_VALIDATION_FLAG_TEST

When this flag is set, the current validation phase is a "test" validation, as in e.g. the CLI 'validate' command, and the transaction will return to the READ state regardless of the validation result. This flag is available in all of the `init()`, `validate()`, and `stop()` callbacks.

CONFD_VALIDATION_FLAG_COMMIT

When this flag is set, all requirements for a commit have been met, i.e. all validation as well as the `write_start` and `prepare` transitions have been successful, and the actual commit will follow. This flag is only available in the `stop()` callback.

```

int confd_register_range_valpoint_cb(struct confd_daemon_ctx *dx,
struct confd_valpoint_cb *vcb, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);

```

A variant of `confd_register_valpoint_cb()` which registers a validation function for a range of key values. The *lower*, *upper*, *numkeys*, *fmt*, and remaining parameters are the same as for `confd_register_range_data_cb()`, see above.

```
int confd_delayed_reply_validation_warn(struct confd_trans_ctx *tctx);
```

This function must be used to return the equivalent of `CONFID_VALIDATION_WARN` when the `validate()` callback returned `CONFID_DELAYED_RESPONSE`. Before calling this function, we must call `confd_trans_seterr()` to provide a string describing the warning.

Errors: `CONFID_ERR_PROTOUSAGE`, `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`

NOTIFICATION STREAMS

The application can generate notifications that are sent via the northbound protocols. Currently `NETCONF` notification streams are supported. The application generates the content for each notification and sends it via a socket to `ConfD`, which in turn manages the stream subscriptions and distributes the notifications accordingly.

A stream always has a "live feed", which is the sequence of new notifications, sent in real time as they are generated. Subscribers may also request "replay" of older, logged notifications if the stream supports this, perhaps transitioning to the live feed when the end of the log is reached. There may be one or more replays active simultaneously with the live feed. `ConfD` forwards replay requests from subscribers to the application via callbacks if the stream supports replay.

Each notification has an associated time stamp, the "event time". This is the time when the event that generated the notification occurred, rather than the time the notification is logged or sent, in case these times differ. The application must pass the event time to `ConfD` when sending a notification, and it is also needed when replaying logged events, see below.

```
int confd_register_notification_stream(struct confd_daemon_ctx
*dx, const struct confd_notification_stream_cbs *ncbs, struct
confd_notification_ctx **nctx);
```

This function registers the notification stream and optionally two callback functions used for the replay functionality. If the stream does not support replay, the callback elements in the `struct confd_notification_stream_cbs` are set to `NULL`. A context pointer is returned via the `**nctx` argument - this must be used by the application for the sending of live notifications via `confd_notification_send()` (see below).

The `confd_notification_stream_cbs` structure is defined as:

```
struct confd_notification_stream_cbs {
    char streamname[MAX_STREAMNAME_LEN];
    int fd;
    int (*get_log_times)(
        struct confd_notification_ctx *nctx);
    int (*replay)(struct confd_notification_ctx *nctx,
        struct confd_datetime *start,
        struct confd_datetime *stop);
    void *cb_opaque; /* private user data */
};
```

The `fd` element must be set to a previously connected worker socket. This socket may be used for multiple notification streams, but not for any of the callback processing described above. Since it is only used for sending data to `ConfD`, there is no need for the application to poll the socket. Note that the control socket must be connected before registration even if the callbacks are not registered.

**Note**

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

The `get_log_times()` callback is called by ConfD to find out a) the creation time of the current log and b) the event time of the last notification aged out of the log, if any. The application provides the times via the `confd_notification_reply_log_times()` function (see below) and returns `CONF_OK`.

The `replay()` callback is called by ConfD to request replay. The `nctx` context pointer must be saved by the application and used when sending the replay notifications via `confd_notification_send()`, as well as for the `confd_notification_replay_complete()` (or `confd_notification_replay_failed()`) call (see below) - the callback should return without waiting for the replay to complete. The pointer references allocated memory, which is freed by the `confd_notification_replay_complete()` (or `confd_notification_replay_failed()`) call.

The times given by `*start` and `*stop` specify the extent of the replay. The start time will always be given and specify a time in the past, however the stop time may be either in the past or in the future or even omitted, i.e. the `stop` argument is `NULL`. This means that the subscriber has requested that the subscription continues indefinitely with the live feed when the logged notifications have been sent.

If the stop time is given:

- The application sends all logged notifications that have an event time later than the start time but not later than the stop time, and then calls `confd_notification_replay_complete()`. Note that if the stop time is in the future when the replay request arrives, this includes notifications logged while the replay is in progress (if any), as long as their event time is not later than the stop time.

If the stop time is *not* given:

- The application sends all logged notifications that have an event time later than the start time, and then calls `confd_notification_replay_complete()`. Note that this includes notifications logged after the request was received (if any).

ConfD will if needed switch the subscriber over to the live feed and then end the subscription when the stop time is reached. The callback may analyze the `start` and `stop` arguments to determine start and stop positions in the log, but if the analysis is postponed until after the callback has returned, the `confd_datetime` structure(s) must be copied by the callback.

The `replay()` callback may optionally select a separate worker socket to be used for the replay notifications. In this case it must call `confd_notification_set_fd()` to indicate which socket should be used.

Note that unlike the callbacks for external data bases and validation, these callbacks do not use a worker socket for the callback processing, and consequently there is no `init()` callback to request one. The callbacks are invoked, and the reply is sent, via the daemon control socket.

The `cb_opaque` element in the `confd_notification_stream_cbs` structure can be used to pass arbitrary data to the callbacks in much the same way as for callpoint and validation point registrations, see the description of the struct `confd_data_cbs` structure above. However since the callbacks are not associated with a transaction, this element is instead made available in the `confd_notification_ctx` structure.


```
int confd_notification_send(struct confd_notification_ctx *nctx, struct
confd_datetime *time, confd_tag_value_t *values, int nvalues);
```

This function is called by the application to send a notification, whether "live" or replay. The *nctx* pointer is provided by ConfD as described above. The *time* argument specifies the event time for the notification. The *values* argument is an array of length *nvalues*, populated with the content of the notification as described for the Tagged Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

For example, a NETCONF notification of the form

```
<ncn:notification
  xmlns:ncn="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <linkUp xmlns="http://example.com/ns/test/1.0">
    <ncn:eventTime>2007-08-17T08:56:05Z</ncn:eventTime>
    <ifIndex>3</ifIndex>
  </linkUp>
</ncn:notification>
```

could be sent with the following code:

```
struct confd_notification_ctx *nctx;
struct confd_datetime event_time = {2007, 8, 17, 8, 56, 5, 0, 0, 0};
confd_tag_value_t notif[3];
int n = 0;

CONFID_SET_TAG_XMLBEGIN(&notif[n], test_linkUp, test__ns); n++;
CONFID_SET_TAG_UINT32(&notif[n], test_ifIndex, 3); n++;
CONFID_SET_TAG_XMLEND(&notif[n], test_linkUp, test__ns); n++;
confd_notification_send(nctx, &event_time, notif, n);
```



Note

While it is possible to use separate threads to send live and replay notifications for a given stream, or to send different streams on a given worker socket, this is not recommended. This is because it involves rather complex synchronization problems that can only be fully solved by the application, in particular in the case where a replay switches over to the live feed.

```
int confd_notification_replay_complete(struct confd_notification_ctx
*nctx);
```

The application calls this function to notify ConfD that the replay is complete, using the *nctx* pointer received in the corresponding `replay()` callback invocation.

```
int confd_notification_replay_failed(struct confd_notification_ctx
*nctx);
```

In case the application fails to complete the replay as requested (e.g. the log gets overwritten while the replay is in progress), the application should call this function *instead* of `confd_notification_replay_complete()`. An error message describing the reason for the failure can be supplied by first calling `confd_notification_seterr()` or `confd_notification_seterr_extended()`, see below. The *nctx* pointer received in the corresponding `replay()` callback invocation is used for both calls.

```
void confd_notification_set_fd(struct confd_notification_ctx *nctx, int
fd);
```

This function may optionally be called by the `replay()` callback to request that the worker socket given by `fd` should be used for the replay. Otherwise the socket specified in the `confd_notification_stream_cbs` at registration will be used.

```
int confd_notification_reply_log_times(struct confd_notification_ctx
*nctx, struct confd_datetime *creation, struct confd_datetime *aged);
```

Reply function for use in the `get_log_times()` callback invocation. If no notifications have been aged out of the log, give NULL for the `aged` argument.

```
void confd_notification_seterr(struct confd_notification_ctx *nctx,
const char *fmt, ...);
```

In some cases the callbacks may be unable to carry out the requested actions, e.g. the capacity for simultaneous replays might be exceeded, and they can then return `CONFD_ERR`. This function allows the callback to associate an error message with the failure. It can also be used to supply an error message before calling `confd_notification_replay_failed()`.

```
void confd_notification_seterr_extended(struct confd_notification_ctx
*nctx, enum confd_errcode code, u_int32_t apptag_ns, u_int32_t
apptag_tag, const char *fmt, ...);
```

This function can be used to provide more structured error information from a notification callback, see the section [EXTENDED ERROR REPORTING](#) in `confd_lib_lib(3)`.

```
int confd_notification_seterr_extended_info(struct
confd_notification_ctx *nctx, enum confd_errcode code, u_int32_t
apptag_ns, u_int32_t apptag_tag, confd_tag_value_t *error_info, int n,
const char *fmt, ...);
```

This function can be used to provide structured error information in the same way as `confd_notification_seterr_extended()`, and additionally provide contents for the NETCONF `<error-info>` element. See the section [EXTENDED ERROR REPORTING](#) in `confd_lib_lib(3)`.

```
int confd_register_snmp_notification(struct confd_daemon_ctx *dx,
int fd, const char *notify_name, const char *ctx_name, struct
confd_notification_ctx **nctx);
```

SNMP notifications can also be sent via the notification framework, however most aspects of the stream concept described above do not apply for SNMP. This function is used to register a worker socket, the `snmpNotifyName` (`notify_name`), and SNMP context (`ctx_name`) to be used for the notifications.

The `fd` parameter must give a previously connected worker socket. This socket may be used for different notifications, but not for any of the callback processing described above. Since it is only used for sending data to ConfD, there is no need for the application to poll the socket. Note that the control socket must be connected before registration, even if none of the callbacks described below are registered.

The context pointer returned via the `**nctx` argument must be used by the application for the subsequent sending of the notifications via `confd_notification_send_snmp()` or `confd_notification_send_snmp_inform()` (see below).

When a notification is sent using one of these functions, it is delivered to the management targets defined for the `snmpNotifyName` in the `snmpNotifyTable` in SNMP-NOTIFICATION-MIB for the

specified SNMP context. If *notify_name* is NULL or the empty string (""), the notification is sent to all management targets. If *ctx_name* is NULL or the empty string (""), the default context ("") is used.



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

```
int confd_notification_send_snmp(struct confd_notification_ctx *nctx,
const char *notification, struct confd_snmp_varbind *varbinds, int
num_vars);
```

Sends the SNMP notification specified by *notification*, without requesting inform-request delivery information. This is equivalent to calling `confd_notification_send_snmp_inform()` (see below) with NULL as the *cb_id* argument. I.e. if the common arguments are the same, the two functions will send the exact same set of traps and inform-requests.

```
int confd_register_notification_snmp_inform_cb(struct confd_daemon_ctx
*dx, const struct confd_notification_snmp_inform_cbs *cb);
```

If we want to receive information about the delivery of SNMP inform-requests, we must register two callbacks for this. The struct `confd_notification_snmp_inform_cbs` is defined as:

```
struct confd_notification_snmp_inform_cbs {
    char cb_id[MAX_CALLPOINT_LEN];
    void (*targets)(struct confd_notification_ctx *nctx,
                    int ref, struct confd_snmp_target *targets,
                    int num_targets);
    void (*result)(struct confd_notification_ctx *nctx,
                  int ref, struct confd_snmp_target *target,
                  int got_response);
    void *cb_opaque; /* private user data */
};
```

The callback identifier *cb_id* can be chosen arbitrarily, it is only used when sending SNMP notifications with `confd_notification_send_snmp_inform()` - however each inform callback registration must use a unique *cb_id*. The callbacks are invoked via the control socket, i.e. the application must poll it and invoke `confd_fd_ready()` when data is available.

When a notification is sent, the `target()` callback will be invoked once with *num_targets* (possibly 0) inform-request targets in the *targets* array, followed by *num_targets* invocations of the `result()` callback, one for each target. The *ref* argument (passed from the `confd_notification_send_snmp_inform()` call) allows for tracking the result of multiple notifications with delivery overlap.



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

```
int confd_notification_send_snmp_inform(struct confd_notification_ctx
*nctx, const char *notification, struct confd_snmp_varbind *varbinds,
int num_vars, const char *cb_id, int ref);
```

Sends the SNMP notification specified by *notification*. If *cb_id* is not NULL, the callbacks registered for *cb_id* will be invoked with the *ref* argument as described above, otherwise no inform-

request delivery information will be provided. The *varbinds* array should be populated with *num_vars* elements as described in the Notifications section of the SNMP Agent chapter in the User Guide.

If *notification* is the empty string, no notification is looked up; instead *varbinds* defines the notification, including the notification id (variable name "snmpTrapOID"). This is especially useful for forwarding a notification which has been received from the SNMP gateway (see `confd_register_notification_sub_snmp_cb()` below).

If *varbinds* does not contain a timestamp (variable name "sysUpTime"), one will be supplied by the agent.

```
void confd_notification_set_snmp_src_addr(struct confd_notification_ctx
*nctx, const struct confd_ip *src_addr);
```

By default, the source address for the SNMP notifications that are sent by the above functions is chosen by the IP stack of the OS. This function may be used to select a specific source address, given by *src_addr*, for the SNMP notifications subsequently sent using the *nctx* context. The default can be restored by calling the function with a *src_addr* where the *af* element is set to `AF_UNSPEC`.

```
int confd_notification_set_snmp_notify_name(struct
confd_notification_ctx *nctx, const char *notify_name);
```

This function can be used to change the `snmpNotifyName` (*notify_name*) for the *nctx* context. The new `snmpNotifyName` is used for notifications sent by subsequent calls to `confd_notification_send_snmp()` and `confd_notification_send_snmp_inform()` that use the *nctx* context.

```
int confd_register_notification_sub_snmp_cb(struct confd_daemon_ctx
*dx, const struct confd_notification_sub_snmp_cb *cb);
```

Registers a callback function to be called when an SNMP notification is received by the SNMP gateway.

The struct `confd_notification_sub_snmp_cb` is defined as:

```
struct confd_notification_sub_snmp_cb {
    char sub_id[MAX_CALLPOINT_LEN];
    int (*recv)(struct confd_notification_ctx *nctx,
                char *notification,
                struct confd_snmp_varbind *varbinds, int num_vars,
                confd_value_t *src_addr, u_int16_t src_port);
    void *cb_opaque; /* private user data */
};
```

The *sub_id* element is the subscription id for the notifications. The `recv()` callback will be called when a notification is received. See the section "Receiving and Forwarding Traps" in the chapter "The SNMP gateway" in the Users Guide.



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

```
int confd_notification_flush(struct confd_notification_ctx *nctx);
```

Notifications are sent asynchronously, i.e. normally without blocking the caller of the send functions described above. This means that in some cases, ConfD's sending of the notifications on the northbound interfaces may lag behind the send calls. If we want to make sure that the notifications have actually

been sent out, e.g. in some shutdown procedure, we can call `confd_notification_flush()`. This function will block until all notifications sent using the given `nctx` context have been fully processed by ConfD. It can be used both for notification streams and for SNMP notifications (however it will not wait for replies to SNMP inform-requests to arrive).

CONF D ACTIONS

The use of action callbacks can be specified either via a `rpc` statement or via a `tailf:action` statement in the YANG data model, see the YANG specification and [tailf_yang_extensions\(5\)](#). In both cases the use of a `tailf:actionpoint` statement specifies that the action is implemented as a callback function. This section describes how such callback functions should be implemented and registered with ConfD.

Unlike the callbacks for data and validation, there is not always a transaction associated with an action callback. However an action is always associated with a user session (NETCONF, CLI, etc), and only one action at a time can be invoked from a given user session. Hence a pointer to the associated struct `confd_user_info` is passed to the callbacks.

The action callback mechanism is also used for command and completion callbacks configured for the CLI, either in a YANG module using `tailf` extension statements, or in a [clispec\(5\)](#). As the parameter structure is significantly different, special callbacks are used for these functions.

```
int confd_register_action_cbs(struct confd_daemon_ctx *dx, const struct
confd_action_cbs *acb);
```

This function registers up to five callback functions, two of which will be called in sequence when an action is invoked. The struct `confd_action_cbs` is defined as:

```
struct confd_action_cbs {
    char actionpoint[MAX_CALLPOINT_LEN];
    int (*init)(struct confd_user_info *uinfo);
    int (*abort)(struct confd_user_info *uinfo);
    int (*action)(struct confd_user_info *uinfo,
                  struct xml_tag *name,
                  confd_hkeypath_t *kp,
                  confd_tag_value_t *params,
                  int nparams);
    int (*command)(struct confd_user_info *uinfo,
                   char *path, int argc, char **argv);
    int (*completion)(struct confd_user_info *uinfo,
                      int cli_style, char *token, int completion_char,
                      confd_hkeypath_t *kp,
                      char *cmdpath, char *cmdparam_id,
                      struct confd_qname *simpleType, char *extra);
    void *cb_opaque; /* private user data */
};
```

The `init()` callback, and at least one of the `action()`, `command()`, and `completion()` callbacks, must be specified. It is in principle possible to use a single "point name" for more than one of these callback types, and have the corresponding callback invoked in each case, but in typical usage we would only register one of the callbacks `action()`, `command()`, and `completion()`. Below, the term "action callback" is used to refer to any of these three.

Similar to the `init()` callback for external data bases, we must in the `init()` callback associate a worker socket with the action. This socket will be used for the invocation of the action callback, which actually carries out the action. Thus in a multi threaded application, actions can be dispatched to different threads.

However note that unlike the callbacks for external data bases and validation, both `init()` and `action` callbacks are registered for each action point (i.e. different action points can have different `init()` callbacks), and there is no `finish()` callback - the action is completed when the action callback returns.

The struct `confd_action_ctx` `actx` element inside the struct `confd_user_info` holds action-specific data, in particular the `t_opaque` element could be used to pass data from the `init()` callback to the action callback, if needed. If the action is associated with a transaction, the `thandle` element is set to the transaction handle, and can be used with a call to `maapi_attach2()` (see [confd_lib_maapi\(3\)](#)). This is the case for all types of action callbacks invoked from the CLI and Web UI, and for the `action()` callback when invoked via `maapi_request_action_th()` (see [confd_lib_maapi\(3\)](#)) - in other cases, this element is -1.

The `cb_opaque` element in the `confd_action_cbs` structure can be used to pass arbitrary data to the callbacks in much the same way as for callpoint and validation point registrations, see the description of the struct `confd_data_cbs` structure above. This element is made available in the `confd_action_ctx` structure.

If the `tailf:opaque` substatement has been used with the `tailf:actionpoint` statement in the data model, the argument string is made available to the callbacks via the `actionpoint_opaque` element in the `confd_action_ctx` structure.



Note

We must call the `confd_register_done()` function when we are done with all registrations for a daemon, see above.

The `action()` callback receives all the parameters pertaining to the action: The `name` argument is a pointer to the action name as defined in the data model, the `kp` argument gives the path through the data model for an action defined via `tailf:action` (it is a NULL pointer for an action defined via `rpc`), and finally the `params` argument is a representation of the inout parameters provided when the action is invoked. The `params` argument is an array of length `nparams`, populated as described for the Tagged Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

The `command()` callback is invoked for CLI callback commands. It must always result in a call of `confd_action_reply_command()`. As the parameters in this case are all in string form, they are passed in the traditional Unix `argc, argv` manner - i.e. `argv` is an array of `argc` pointers to NUL-terminated strings plus a final NULL pointer element, and `argv[0]` is the name of the command. Additionally the full path of the command is available via the `path` argument.

The `completion()` callback is invoked for CLI completion and information. It must result in a call of `confd_action_reply_completion()`, except for the case when the callback is invoked via a `tailf:cli-custom-range-enumerator` statement in the data model (see below). The `cli_style` argument gives the style of the CLI session as a character: 'J', 'C', or 'I'. The `token` argument is a NUL-terminated string giving the parameter of the CLI command line that the callback invocation pertains to, and `completion_char` is the character that the user typed, i.e. TAB ('\t'), SPACE (' '), or '?'. If the callback pertains to a data model element, `kp` identifies that element, otherwise it is NULL. The `cmdpath` is a NUL-terminated string giving the full path of the command. If a `cli-completion-id` is specified in the YANG module, or a `completionId` is specified in the `clispec`, it is given as a NUL-terminated string via `cmdparam_id`, otherwise this argument is NULL. If the invocation pertains to an element that has a type definition, the `simpleType` argument identifies the type with namespace and type name, otherwise it is NULL. The `extra` argument is currently unused (always NULL).

When `completion()` is invoked via a `tailf:cli-custom-range-enumerator` statement in the data model, it is a request to provide possible key values for creation of an

entry in a list with a custom range specification. The callback must in this case result in a call of `confd_action_reply_range_enum()`. Refer to the `cli/range_create` example in the bundled examples collection to see an implementation of such a callback.

The action callbacks must return `CONF_OK`, `CONF_ERR`, or `CONF_DELAYED_RESPONSE`. `CONF_DELAYED_RESPONSE` implies that the application must later reply asynchronously.

The optional `abort()` callback is called whenever an action is aborted, e.g. when a user invokes an action from one of the northbound agents and aborts it before it has completed. The `abort()` callback will be invoked on the control socket. It is the responsibility of the `abort()` callback to make sure that the pending reply from the action callback is sent. This is required to allow the worker socket to be used for further queries. There are several possible ways for an application to support aborting. E.g. the application can return `CONF_DELAYED_RESPONSE` from the action callback. Then, when the `abort()` callback is called, it can terminate the executing action and use e.g. `confd_action_delayed_reply_error()`. Alternatively an application can use threads where the action callback is executed in a separate thread. In this case the `abort()` callback could inform the thread executing the action that it should be terminated, and that thread can just return from the action callback.

```
int confd_register_range_action_cbs(struct confd_daemon_ctx *dx,
const struct confd_action_cbs *acb, const confd_value_t *lower, const
confd_value_t *upper, int numkeys, const char *fmt, ...);
```

A variant of `confd_register_action_cbs()` which registers action callbacks for a range of key values. The `lower`, `upper`, `numkeys`, `fmt`, and remaining parameters are the same as for `confd_register_range_data_cb()`, see above.



Note

This function can not be used for registration of the `command()` or `completion()` callbacks - only actions specified in the data model are invoked via a keypath that can be used for selection of the corresponding callbacks.

```
void confd_action_set_fd(struct confd_user_info *uinfo, int sock);
```

Associate a worker socket with the action. This function must be called in the `init()` callback - a typical implementation of an `init()` callback looks as:

```
static int init_action(struct confd_user_info *uinfo)
{
    confd_action_set_fd(uinfo, workersock);
    return CONF_OK;
}
```

```
int confd_action_reply_values(struct confd_user_info *uinfo,
confd_tag_value_t *values, int nvalues);
```

If the action definition specifies that the action should return data, it must invoke this function in response to the `action()` callback. The `values` argument points to an array of length `nvalues`, populated with the output parameters in the same way as the `params` array above.



Note

This function must only be called for an `action()` callback.

```
int confd_action_reply_command(struct confd_user_info *uinfo, char
**values, int nvalues);
```

If a CLI callback command should return data, it must invoke this function in response to the `command()` callback. The *values* argument points to an array of length *nvalues*, populated with pointers to NUL-terminated strings.



Note

This function must only be called for a `command()` callback.

```
int confd_action_reply_rewrite(struct confd_user_info *uinfo, char
**values, int nvalues, char **unhides, int nunhides);
```

This function can be called instead of `confd_action_reply_command()` as a response to a show path rewrite callback invocation. The *values* argument points to an array of length *nvalues*, populated with pointers to NUL-terminated strings representing the tokens of the new path. The *unhides* argument points to an array of length *nunhides*, populated with pointers to NUL-terminated strings representing hide groups to temporarily unhide during evaluation of the show command.



Note

This function must only be called for a `command()` callback.

```
int confd_action_reply_rewrite2(struct confd_user_info *uinfo,
char **values, int nvalues, char **unhides, int nunhides, struct
confd_rewrite_select **selects, int nselects);
```

This function can be called instead of `confd_action_reply_command()` as a response to a show path rewrite callback invocation. The *values* argument points to an array of length *nvalues*, populated with pointers to NUL-terminated strings representing the tokens of the new path. The *unhides* argument points to an array of length *nunhides*, populated with pointers to NUL-terminated strings representing hide groups to temporarily unhide during evaluation of the show command. The *selects* argument points to an array of length *nselects*, populated with pointers to `confd_rewrite_select` structs representing additional select targets.



Note

This function must only be called for a `command()` callback.

```
int confd_action_reply_completion(struct confd_user_info *uinfo, struct
confd_completion_value *values, int nvalues);
```

This function must normally be called in response to the `completion()` callback. The *values* argument points to an *nvalues* long array of `confd_completion_value` elements:

```
enum confd_completion_type {
    CONFD_COMPLETION,
    CONFD_COMPLETION_INFO,
    CONFD_COMPLETION_DESC,
    CONFD_COMPLETION_DEFAULT
};

struct confd_completion_value {
    enum confd_completion_type type;
    char *value;
    char *extra;
};
```


For a completion alternative, `type` is set to `CONF_D_COMPLETION`, `value` gives the alternative as a NUL-terminated string, and `extra` gives explanatory text as a NUL-terminated string - if there is no such text, `extra` is set to `NULL`. For "info" or "desc" elements, `type` is set to `CONF_D_COMPLETION_INFO` or `CONF_D_COMPLETION_DESC`, respectively, and `value` gives the text as a NUL-terminated string (the `extra` element is ignored).

In order to fallback to the normal completion behavior, `type` should be set to `CONF_D_COMPLETION_DEFAULT`. `CONF_D_COMPLETION_DEFAULT` cannot be combined with the other completion types, implying the `values` array always must have length 1 which is indicated by `nvalues` setting.

**Note**

This function must only be called for a `completion()` callback.

```
int confd_action_reply_range_enum(struct confd_user_info *uinfo, char
**values, int keysize, int nkeys);
```

This function must be called in response to the `completion()` callback when it is invoked via a `tailf:cli-custom-range-enumerator` statement in the data model. The `values` argument points to a `keysize * nkeys` long array of strings giving the possible key values, where `keysize` is the number of keys for the list in the data model and `nkeys` is the number of list entries for which keys are provided. I.e. the array gives `entry1-key1`, `entry1-key2`, ..., `entry2-key1`, `entry2-key2`, ... and so on. See the `cli/range_create` example in the bundled examples collection for details.

**Note**

This function must only be called for a `completion()` callback.

```
void confd_action_seterr(struct confd_user_info *uinfo, const char
*fmt, ...);
```

If action callback encounters fatal problems that can not be expressed via the reply function, it may call this function with an appropriate message and return `CONF_D_ERR` instead of `CONF_D_OK`.

```
void confd_action_seterr_extended(struct confd_user_info *uinfo, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);
```

This function can be used to provide more structured error information from an action callback, see the section [EXTENDED ERROR REPORTING](#) in `confd_lib_lib(3)`.

```
int confd_action_seterr_extended_info(struct confd_user_info *uinfo,
enum confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);
```

This function can be used to provide structured error information in the same way as `confd_action_seterr_extended()`, and additionally provide contents for the NETCONF <error-info> element. See the section [EXTENDED ERROR REPORTING](#) in `confd_lib_lib(3)`.

```
int confd_action_delayed_reply_ok(struct confd_user_info *uinfo);
```

```
int confd_action_delayed_reply_error(struct confd_user_info *uinfo,
const char *errstr);
```

If we use the `CONFID_DELAYED_RESPONSE` as a return value from the action callback, we must later asynchronously reply. If we use one of the `confd_action_reply_xxx()` functions, this is a complete reply. Otherwise we must use the `confd_action_delayed_reply_ok()` function to signal success, or the `confd_action_delayed_reply_error()` function to signal an error.

```
int confd_action_set_timeout(struct confd_user_info *uinfo, int
timeout_secs);
```

Some action callbacks may require a significantly longer execution time than others, and this time may not even be possible to determine statically (e.g. a file download). In such cases the `/confdConfig/capi/queryTimeout` setting in `confd.conf` (see above) may be insufficient, and this function can be used to extend (or shorten) the timeout for the current callback invocation. The timeout is given in seconds from the point in time when the function is called.

Examples on how to work with actions are available in the User Guide and in the bundled examples collection.

AUTHENTICATION CALLBACK

We can register a callback with ConfD's AAA subsystem, to be invoked whenever AAA has completed processing of an authentication attempt. In the case where the authentication was otherwise successful, the callback can still cause it to be rejected. This can be used to implement specific access policies, as an alternative to using PAM or "External" authentication for this purpose. The callback will only be invoked if it is both enabled via `/confdConfig/aaa/authenticationCallback/enabled` in `confd.conf` (see [confd.conf\(5\)](#)) and registered as described here.



Note

If the callback is enabled in `confd.conf` but not registered, or invocation keeps failing for some reason, *all* authentication attempts will fail.



Note

This callback can not be used to actually *perform* the authentication. If we want to implement the authentication outside of ConfD, we need to use PAM or "External" authentication, see the [AAA chapter in the User Guide](#).

```
int confd_register_auth_cb(struct confd_daemon_ctx *dx, const struct
confd_auth_cb *acb);
```

Registers the authentication callback. The struct `confd_auth_cb` is defined as:

```
struct confd_auth_cb {
    int (*auth)(struct confd_auth_ctx *actx);
};
```

The `auth()` callback is invoked with a pointer to an authentication context that provides information about the result of the authentication so far. The callback must return `CONFID_OK` or `CONFID_ERR`, see below. The struct `confd_auth_ctx` is defined as:

```
struct confd_auth_ctx {
    struct confd_user_info *uinfo;
    char *method;
    int success;
    union {
```

```

        struct {          /* if success */
            int ngroups;
            char **groups;
        } succ;
        struct {          /* if !success */
            int logno;     /* number from confd_logsyms.h */
            char *reason;
        } fail;
    } ainfo;
    /* ConfD internal fields */
    char *errstr;
};

```

The `uinfo` element points to a struct `confd_user_info` with details about the user logging in, specifically user name, password (if used), source IP address, context, and protocol. Note that the user session does not actually exist at this point, even if the AAA authentication was successful - it will only be created if the callback accepts the authentication, hence e.g. the `userid` element is always 0.

The method string gives the authentication method used, as follows:

"password"	Password authentication. This generic term is used if the authentication failed.
"local", "pam", "external"	Password authentication. On successful authentication, the specific method that succeeded is given. See the AAA chapter in the User Guide for an explanation of these methods.
"publickey"	Public key authentication via the internal SSH server.
Other	Authentication with an unknown or unsupported method with this name was attempted via the internal SSH server.

If `success` is non-zero, the AAA authentication succeeded, and `groups` is an array of length `ngroups` that gives the groups that will be assigned to the user at login. If the callback returns `CONFD_OK`, the complete authentication succeeds and the user is logged in. If it returns `CONFD_ERR` (or an invalid return value), the authentication fails.

If `success` is zero, the AAA authentication failed, with the reason given by `logno` (one of `CONFD_BAD_LOCAL_PASS`, `CONFD_NO_SUCH_LOCAL_USER`, or `CONFD_SSH_NO_LOGIN`) and the explanatory string `reason`. This invocation is only for informational purposes - the callback return value has no effect on the authentication, and should normally be `CONFD_OK`.

```

void confd_auth_seterr(struct confd_auth_ctx *actx, const char
*fmt, ...);

```

This function can be used to provide a text message when the callback returns `CONFD_ERR`. If used when rejecting a successful authentication, the message will be logged in ConfD's audit log (otherwise a generic "rejected by application callback" message is logged).

AUTHORIZATION CALLBACKS

We can register two authorization callbacks with ConfD's AAA subsystem. These will be invoked when the northbound agents check that a command or a data access is allowed by the AAA access rules. The callbacks can partially or completely replace the access checks done within the AAA subsystem, and they may accept or reject the access. Typically many access checks are done during the processing of commands etc, and using these callbacks can thus have a significant performance impact. Unless it is a requirement to query an external authorization mechanism, it is far better to only configure access rules in the AAA data model (see the [AAA chapter in the User Guide](#)).

The callbacks will only be invoked if they are both enabled via `/confdConfig/aaa/authorization/callback/enabled` in `confd.conf` (see [confd.conf\(5\)](#)) and registered as described here.



Note

If the callbacks are enabled in `confd.conf` but no registration has been done, or if invocation keeps failing for some reason, *all* access checks will be rejected.

```
int confd_register_authorization_cb(struct confd_daemon_ctx *dx, const
struct confd_authorization_cbs *acb);
```

Registers the authorization callbacks. The struct `confd_authorization_cbs` is defined as:

```
struct confd_authorization_cbs {
    int cmd_filter;
    int data_filter;
    int (*chk_cmd_access)(struct confd_authorization_ctx *actx,
                        char **cmdtokens, int ntokens, int cmdop);
    int (*chk_data_access)(struct confd_authorization_ctx *actx,
                        u_int32_t hashed_ns, confd_hkeypath_t *hkp,
                        int dataop, int how);
};
```

Both callbacks are optional, i.e. we can set the function pointer in struct `confd_authorization_cbs` to NULL if we don't want the corresponding callback invocation. In this case the AAA subsystem will handle the access check as if the callback was registered, but always replied with `CONFID_ACCESS_RESULT_DEFAULT` (see below).

The `cmd_filter` and `data_filter` elements can be used to prevent access checks from causing invocation of a callback even though it is registered. If we do not want any filtering, they must be set to zero. The value is a bitmask obtained by ORing together values: For `cmd_filter`, we can use the possible values for `cmdop` (see below), preventing the corresponding invocations of `chk_cmd_access()`. For `data_filter`, we can use the possible values for `dataop` and `how` (see below), preventing the corresponding invocation of `chk_data_access()`. If the callback invocation is prevented by filtering, the AAA subsystem will handle the access check as if the callback had replied with `CONFID_ACCESS_RESULT_CONTINUE` (see below).

Both callbacks are invoked with a pointer to an authorization context that provides information about the user session that the access check pertains to, and the group list for that session. The struct `confd_authorization_ctx` is defined as:

```
struct confd_authorization_ctx {
    struct confd_user_info *uinfo;
    int ngroups;
    char **groups;
    struct confd_daemon_ctx *dx;
    /* ConfD internal fields */
    int result;
    int query_ref;
};
```

`chk_cmd_access()`

This callback is invoked for command authorization, i.e. it corresponds to the rules under `/aaa/authorization/cmdrules` in the AAA data model. `cmdtokens` is an array of `ntokens` NUL-terminated strings representing the command to be checked, corresponding to the command leaf

in the *cmdrule* list. If */confdConfig/cli/modeInfoInAAA* is enabled in *confd.conf* (see [confd.conf\(5\)](#)), mode names will be prepended in the *cmdtokens* array. The *cmdop* parameter gives the operation, corresponding to the *ops* leaf in the *cmdrule* list. The possible values for *cmdop* are:

CONF_D_ACCESS_OP_READ

Read access. The CLI will use this during command completion, to filter out alternatives that are disallowed by AAA.

CONF_D_ACCESS_OP_EXECUTE

Execute access. This is used when a command is about to be executed.



Note

This callback may be invoked with *actx->uinfo == NULL*, meaning that no user session has been established for the user yet. This will occur e.g. when the CLI checks whether a user attempting to log in is allowed to (implicitly) execute the command "request system logout user" (J-CLI) or "logout" (C/I-CLI) when the maximum number of sessions has already been reached (if allowed, the CLI will ask whether the user wants to terminate one of the existing sessions).

chk_data_access()

This callback is invoked for data authorization, i.e. it corresponds to the rules under */aaa/authorization/datarules* in the AAA data model. *hashed_ns* and *hkp* give the namespace and hkeypath of the data node to be checked, corresponding to the namespace and keypath leafs in the *datarule* list. The *hkp* parameter may be *NULL*, which means that access to the entire namespace given by *hashed_ns* is requested. When a hkeypath is provided, some key elements in the path may be without key values (i.e. *hkp->v[n][0].type == C_NOEXISTS*). This indicates "wildcard" keys, used for CLI tab completion when keys are not fully specified. The *dataop* parameter gives the operation, corresponding to the *ops* leaf in the *datarule* list. The possible values for *dataop* are:

CONF_D_ACCESS_OP_READ

Read access.

CONF_D_ACCESS_OP_EXECUTE

Execute access.

CONF_D_ACCESS_OP_CREATE

Create access.

CONF_D_ACCESS_OP_UPDATE

Update access.

CONF_D_ACCESS_OP_DELETE

Delete access.

CONF_D_ACCESS_OP_WRITE

Write access. This is used when the specific write operation (create/update/delete) isn't known yet, e.g. in CLI command completion or processing of a NETCONF **edit-config**.

The *how* parameter is one of:

CONF_D_ACCESS_CHK_INTERMEDIATE

Access to the given data node *or* its descendants is requested. This is used e.g. in CLI command completion or processing of a NETCONF **edit-config**.

CONF_D_ACCESS_CHK_FINAL

Access to the specific data node is requested.

```
int confd_access_reply_result(struct confd_authorization_ctx *actx, int result);
```

The callbacks must call this function to report the result of the access check to ConfD, and should normally return `CONF_D_OK`. If any other value is returned, it will cause the access check to be rejected. The *actx* parameter is the pointer to the authorization context passed in the callback invocation, and *result* must be one of:

`CONF_D_ACCESS_RESULT_ACCEPT`

The access is allowed. This is a "final verdict", analogous to a "full match" when the AAA rules are used.

`CONF_D_ACCESS_RESULT_REJECT`

The access is denied.

`CONF_D_ACCESS_RESULT_CONTINUE`

The access is allowed "so far". I.e. access to sub-elements is not necessarily allowed. This result is mainly useful when `chk_cmd_access()` is called with `cmdop == CONF_D_ACCESS_OP_READ` or `chk_data_access()` is called with `how == CONF_D_ACCESS_CHK_INTERMEDIATE`.

`CONF_D_ACCESS_RESULT_DEFAULT`

The request should be handled according to the rules configured in the AAA data model.

```
int confd_authorization_set_timeout(struct confd_authorization_ctx
*actx, int timeout_secs);
```

The authorization callbacks are invoked on the daemon control socket, and as such are expected to complete quickly, within the timeout specified for `/confdConfig/capi/newSessionTimeout`. However in case they send requests to a remote server, and such a request needs to be retried, this function can be used to extend the timeout for the current callback invocation. The timeout is given in seconds from the point in time when the function is called.

ERROR FORMATTING CALLBACK

It is possible to register a callback function to generate customized error messages for ConfD's internally generated errors. All the customizable errors are defined with a type and a code in the XML document `$CONF_D_DIR/src/confd/errors/errcode.xml` in the ConfD release. To use this functionality, the application must `#include` the file `confd_errcode.h`, which defines C constants for the types and codes.

```
int confd_register_error_cb(struct confd_daemon_ctx *dx, const struct
confd_error_cb *ecb);
```

Registers the error formatting callback. The struct `confd_error_cb` is defined as:

```
struct confd_error_cb {
    int error_types;
    void (*format_error)(struct confd_user_info *uinfo,
                        struct confd_errinfo *errinfo,
                        char *default_msg);
};
```

The `error_types` element is the logical OR of the error types that the callback should handle. An application daemon can only register one error formatting callback, and only one daemon can register for each error type. The available types are:

`CONF_D_ERRTYPE_VALIDATION`

Errors detected by ConfD's internal semantic validation of the data model constraints, e.g. mandatory elements that are unset, dangling references, etc. The codes for this type are the `confd_errno` values corresponding to the validation errors, as resulting e.g. from a call

to `maapi_apply_trans()` (see [confd_lib_maapi\(3\)](#)). I.e. `CONFD_ERR_NOTSET`, `CONFD_ERR_BAD_KEYREF`, etc - see the 'id' attribute in `errcode.xml`.

`CONFD_ERRTYPE_BAD_VALUE`

Type errors, i.e. errors generated when an invalid value is given for a leaf in the data model. The codes for this type are defined in `confd_errcode.h` as `CONFD_BAD_VALUE_XXX`, where "XXX" is the all-uppercase form of the code name given in `errcode.xml`.

`CONFD_ERRTYPE_CLI`

CLI-specific errors. The codes for this type are defined in `confd_errcode.h` as `CONFD_CLI_XXX` in the same way as for `CONFD_ERRTYPE_BAD_VALUE`.

`CONFD_ERRTYPE_MISC`

Miscellaneous errors, which do not fit into the other categories. The codes for this type are defined in `confd_errcode.h` as `CONFD_MISC_XXX` in the same way as for `CONFD_ERRTYPE_BAD_VALUE`.

The `format_error()` callback is invoked with a pointer to a struct `confd_errinfo`, which gives the error type and type-specific structured information about the details of the error. It is defined as:

```
struct confd_errinfo {
    int type; /* CONF_ERRTYPE_XXX */
    union {
        struct confd_errinfo_validation validation;
        struct confd_errinfo_bad_value bad_value;
        struct confd_errinfo_cli cli;
        struct confd_errinfo_misc misc;
    } info;
};
```

For `CONF_ERRTYPE_VALIDATION`, the struct `confd_errinfo_validation` validation gives the detailed information, using an info union that has a specific struct member for each code:

```
struct confd_errinfo_validation {
    int code; /* CONF_ERR_NOTSET, CONF_ERR_TOO_FEW_ELEMS, ... */
    union {
        struct {
            /* the element given by kp is not set */
            confd_hkeypath_t *kp;
        } notset;
        struct {
            /* kp has n instances, must be at least min */
            confd_hkeypath_t *kp;
            int n, min;
        } too_few_elems;
        struct {
            /* kp has n instances, must be at most max */
            confd_hkeypath_t *kp;
            int n, max;
        } too_many_elems;
        struct {
            /* the elements given by kps1 have the same set
               of values vals as the elements given by kps2
               (kps1, kps2, and vals point to n_elems long arrays) */
            int n_elems;
            confd_hkeypath_t *kps1;
            confd_hkeypath_t *kps2;
            confd_value_t *vals;
        } non_unique;
        struct {
            /* the element given by kp references
```

```

        the non-existing element given by ref
        Note: 'ref' may be NULL or have key elements without values
        (ref->v[n][0].type == C_NOEXISTS) if it cannot be instantiated */
        confd_hkeypath_t *kp;
        confd_hkeypath_t *ref;
    } bad_keyref;
    struct {
        /* the mandatory 'choice' statement choice in the
           container kp does not have a selected 'case' */
        confd_value_t *choice;
        confd_hkeypath_t *kp;
    } unset_choice;
    struct {
        /* the 'must' expression expr for element kp is not satisfied
           - error_message and error_app_tag are NULL if not given
           in the 'must'; val points to the value of the element if it
           has one, otherwise it is NULL */
        char *expr;
        confd_hkeypath_t *kp;
        char *error_message;
        char *error_app_tag;
        confd_value_t *val;
    } must_failed;
    struct {
        /* the element kp has the instance-identifier value instance,
           which doesn't exist, but require-instance is 'true' */
        confd_hkeypath_t *kp;
        confd_hkeypath_t *instance;
    } missing_instance;
    struct {
        /* the element kp has the instance-identifier value instance,
           which doesn't conform to the specified path filters */
        confd_hkeypath_t *kp;
        confd_hkeypath_t *instance;
    } invalid_instance;
    struct {
        /* the expression for a configuration policy rule evaluated to
           'false' - error_message is the associated error message */
        char *error_message;
    } policy_failed;
    struct {
        /* the XPath expression expr, for the configuration policy
           rule with key name, could not be compiled due to msg */
        char *name;
        char *expr;
        char *msg;
    } policy_compilation_failed;
    struct {
        /* the expression expr, for the configuration policy rule
           with key name, failed XPath evaluation due to msg */
        char *name;
        char *expr;
        char *msg;
    } policy_evaluation_failed;
    } info;
    int test; /* 1 if 'validate', 0 if 'commit' */
    struct confd_trans_ctx *tctx; /* only valid for duration of callback */
};

```

The member structs are named as the `confd_errno` values that are used for the code elements, i.e. notset for `CONFD_ERR_NOTSET`, etc. For this error type, the callback also has full information about the transaction that failed validation via the `struct confd_trans_ctx *tctx` element - it

is even possible to use `maapi_attach()` (see [confd_lib_maapi\(3\)](#)) to attach to the transaction and read arbitrary data from it, in case the data directly related to the error (as given in the code-specific struct) is not sufficient.

For the other error types, the corresponding `confd_errinfo_xxx` struct gives the code and an array with the parameters for the default error message, as defined by the `<fmt>` element in `errcode.xml`:

```
enum confd_errinfo_ptype {
    CONFD_ERRINFO_KEYPATH,
    CONFD_ERRINFO_STRING
};

struct confd_errinfo_param {
    enum confd_errinfo_ptype type;
    union {
        confd_hkeypath_t *kp;
        char *str;
    } val;
};

struct confd_errinfo_bad_value {
    int code;
    int n_params;
    struct confd_errinfo_param *params;
};
```

The parameters in the `params` array are given in the order they appear in the `<fmt>` specification. Parameters that are specified as `{path}` have `params[n].type` set to `CONFD_ERRINFO_KEYPATH`, and are represented as a `confd_hkeypath_t` that can be accessed via `params[n].val.kp`. All other parameters are represented as strings, i.e. `params[n].type` is `CONFD_ERRINFO_STR` and the string value can be accessed via `params[n].val.str`. The struct `confd_errinfo_cli` and struct `confd_errinfo_misc` union members have the same form as struct `confd_errinfo_bad_value` shown above.

Finally, the `default_msg` callback parameter gives the default error message that will be reported to the user if the `format_error()` function does not generate a replacement.

```
void confd_error_seterr(struct confd_user_info *uinfo, const char
*fmt, ...);
```

This function must be called by `format_error()` to provide a replacement of the default error message. If `format_error()` returns without calling `confd_error_seterr()`, the default message will be used.

Here is an example that targets a specific validation error for a specific element in the data model. For this case only, it replaces ConfD's internally generated messages of the form:

```
"too many 'protocol bgp', 2 configured, at most 1 must be configured"
```

with

```
"Only 1 bgp instance is supported, cannot define 2"
```

```
#include <confd_lib.h>
#include <confd_dp.h>
#include <confd_errcode.h>
.
.
int main(int argc, char **argv)
{
```

```

    struct confd_error_cb ecb;
    .
    .
    memset(&ecb, 0, sizeof(ecb));
    ecb.error_types = CONFD_ERRTYPE_VALIDATION;
    ecb.format_error = format_error;
    if (confd_register_error_cb(dctx, &ecb) != CONFD_OK)
        confd_fatal("Couldn't register error callback\n");
    .
}

static void format_error(struct confd_user_info *uinfo,
                        struct confd_errinfo *errinfo,
                        char *default_msg)
{
    struct confd_errinfo_validation *err;
    confd_hkeypath_t *kp;

    err = &errinfo->info.validation;
    if (err->code == CONFD_ERR_TOO_MANY_ELEMS) {
        kp = err->info.too_many_elems.kp;
        if (CONFD_GET_XMLTAG(&kp->v[0][0]) == myns_bgp &&
            CONFD_GET_XMLTAG(&kp->v[1][0]) == myns_protocol) {
            confd_error_seterr(uinfo,
                              "Only %d bgp instance is supported, "
                              "cannot define %d",
                              err->info.too_many_elems.max,
                              err->info.too_many_elems.n);
        }
    }
}

```

The CLI-specific "Aborted: " prefix is not included in the message for this error type - if we wanted to replace that too, we could include the `CONFD_ERRTYPE_CLI` error type in the registration and process the `CONFD_CLI_COMMAND_ABORTED` error code for this type, see `errcode.xml`.

SEE ALSO

`confd.conf(5)` - ConfD daemon configuration file format

The ConfD User Guide

Name

confd_lib_events — library for subscribing to NSO event notifications

Synopsis

```
#include <confd_lib.h> #include <confd_events.h>

int confd_notifications_connect(int sock, const struct sockaddr* srv,
int srv_sz, int mask);

int confd_notifications_connect2(int sock, const struct sockaddr* srv,
int srv_sz, int mask, struct confd_notifications_data *data);

int confd_read_notification(int sock, struct confd_notification *n);

void confd_free_notification(struct confd_notification *n);

int confd_diff_notification_done(int sock, struct confd_trans_ctx
*tctx);

int confd_sync_audit_notification(int sock, int usid);

int confd_sync_ha_notification(int sock);
```

LIBRARY

NSO Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to NSO and subscribe to certain events generated by NSO. The API to receive events from NSO is a socket based API whereby the application connects to NSO and receives events on a socket. See also the [Notifications](#) chapter in the User Guide. The program misc/notifications/confd_notifications.c in the examples collection illustrates subscription and processing for all these events, and can also be used standalone in a development environment to monitor NSO events.

EVENTS

The following events can be subscribed to:

CONFID_NOTIF_AUDIT

All audit log events are sent from ConfD on the event notification socket.

CONFID_NOTIF_AUDIT_SYNC

This flag modifies the behavior of a subscription for the CONFID_NOTIF_AUDIT event - it has no effect unless CONFID_NOTIF_AUDIT is also present. If this flag is present, ConfD will stop processing in the user session that causes an audit notification to be sent, and continue processing in that user session only after all subscribers with this flag have called confd_sync_audit_notification().

CONFID_NOTIF_DAEMON

All log events that also goes to the /confdConf/logs/confdLog log are sent from ConfD on the event notification socket.

CONFD_NOTIF_NETCONF

All log events that also goes to the `/confdConf/logs/netconfLog` log are sent from ConfD on the event notification socket.

CONFD_NOTIF_DEVEL

All log events that also goes to the `/confdConf/logs/developerLog` log are sent from ConfD on the event notification socket.

CONFD_NOTIF_TAKEOVER_SYSLOG

If this flag is present, ConfD will stop syslogging. The idea behind the flag is that we want to configure syslogging for ConfD in order to let ConfD log its startup sequence. Once ConfD is started we wish to subsume the syslogging done by ConfD. Typical applications that use this flag want to pick up all log messages, reformat them and use some local logging method.

Once all subscriber sockets with this flag set are closed, ConfD will resume to syslog.

CONFD_NOTIF_COMMIT_SIMPLE

An event indicating that a user has somehow modified the configuration.

CONFD_NOTIF_COMMIT_DIFF

An event indicating that a user has somehow modified the configuration. The main difference between this event and the abovementioned `CONFD_NOTIF_COMMIT_SIMPLE` is that this event is synchronous, i.e. the entire transaction hangs until we have explicitly called `confd_diff_notification_done()`. The purpose of this event is to give the applications a chance to read the configuration diffs from the transaction before it finishes. A user subscribing to this event can use MAAPI to attach (`maapi_attach()`) to the running transaction and use `maapi_diff_iterate()` to iterate through the diff. This feature can also be used to produce a complete audit trail of who changed what and when in the system. It is up to the application to format that audit trail.

CONFD_NOTIF_COMMIT_FAILED

This event is generated when a data provider fails in its commit callback. ConfD executes a two-phase commit procedure towards all data providers when committing transactions. When a provider fails in commit, the system is in an unknown state. See [confd_lib_maapi\(3\)](#) and the function `maapi_get_running_db_state()`. If the provider is "external", the name of failing daemon is provided. If the provider is another NETCONF agent, the IP address and port of that agent is provided.

CONFD_NOTIF_CONFIRMED_COMMIT

This event is generated when a user has started a confirmed commit, when a confirming commit is issued, or when a confirmed commit is aborted; represented by enum `confd_confirmed_commit_type`.

For a confirmed commit, the timeout value is also present in the notification.

CONFD_NOTIF_COMMIT_PROGRESS

This event provides progress information about the commit of a transaction, i.e. the same information that is reported when the **commit | details** CLI command is used. The application receives a struct `confd_progress_notification` which gives details for the specific transaction along with the progress information, see `confd_events.h`.

CONFD_NOTIF_USER_SESSION

An event related to user sessions. There are 6 different user session related event types, defined in enum `confd_user_sess_type`: session starts/stops, session locks/unlocks database, session starts/stop database transaction.

CONFD_NOTIF_HA_INFO

An event related to ConfDs perception of the current cluster configuration.

CONFID_NOTIF_HA_INFO_SYNC

This flag modifies the behavior of a subscription for the CONFID_NOTIF_HA_INFO event - it has no effect unless CONFID_NOTIF_HA_INFO is also present. If this flag is present, ConfD will stop all HA processing, and continue only after all subscribers with this flag have called `confd_sync_ha_notification()`.

CONFID_NOTIF_SUBAGENT_INFO

Only sent if ConfD runs as a master agent with subagents enabled. This event is sent when the subagent connection is lost or reestablished. There are two event types, defined in enum `confd_subagent_info_type`: subagent up and subagent down.

CONFID_NOTIF_SNMPA

This event is generated whenever an SNMP pdu is processed by ConfD. The application receives a struct `confd_snmpa_notification` structure. The structure contains a series of fields describing the sent or received SNMP pdu. It contains a list of all varbinds in the pdu.

Each varbind contains a `confd_value_t` with the string representation of the SNMP value. Thus the type of the value in a varbind is always `C_BUF`. See `confd_events.h` include file for the details of the received structure.



Note

This event may allocate memory dynamically inside the struct `confd_notification`, thus we must always call `confd_free_notification()` after receiving and processing this event.

CONFID_NOTIF_FORWARD_INFO

This event is generated whenever ConfD forwards (proxies) a northbound agent.

CONFID_NOTIF_UPGRADE_EVENT

This event is generated for the different phases of an in-service upgrade, i.e. when the data model is upgraded while ConfD is running. The application receives a struct `confd_upgrade_notification` where the enum `confd_upgrade_event_type` event gives the specific upgrade event, see `confd_events.h`. The events correspond to the invocation of the MAAPI functions that drive the upgrade, see [confd_lib_maapi\(3\)](#).

CONFID_NOTIF_HEARTBEAT

This event can be used by applications that wish to monitor the health and liveness of ConfD itself. It needs to be requested through a call to `confd_notifications_connect2()`, where the required `heartbeat_interval` can be provided via the *struct* `confd_notifications_data` parameter. ConfD will continuously generate heartbeat events on the notification socket. If ConfD fails to do so, ConfD is hung, or prevented from getting the CPU time required to send the event. The timeout interval is measured in milliseconds. Recommended value is 10000 milliseconds to cater for truly high load situations. Values less than 1000 are changed to 1000.

CONFID_NOTIF_HEALTH_CHECK

This event is similar to CONFID_NOTIF_HEARTBEAT, in that it can be used by applications that wish to monitor the health and liveness of ConfD itself. However while CONFID_NOTIF_HEARTBEAT will be generated as long as ConfD is not completely hung, CONFID_NOTIF_HEALTH_CHECK will only be generated after a basic liveness check of the different ConfD subsystems has completed successfully. This event also needs to be requested through a call to `confd_notifications_connect2()`, where the required `health_check_interval` can be provided via the *struct* `confd_notifications_data` parameter. Since the event generation incurs more processing than CONFID_NOTIF_HEARTBEAT, a longer interval than 10000 milliseconds is recommended, but in particular the application must be prepared for the actual interval

to be significantly longer than the requested one in high load situations. Values less than 1000 are changed to 1000.

CONFD_NOTIF_REOPEN_LOGS

This event indicates that NSO will close and reopen its log files, i.e. that **ncs --reload** or **maapi_reopen_logs()** (e.g. via **ncs_cmd -c reopen_logs**) has been used.

NCS_NOTIF_PACKAGE_RELOAD

This event is generated whenever NCS has completed a package reload.

NCS_NOTIF_CQ_PROGRESS

This event is generated to report the progress of commit queue entries.

The application receives a struct `ncs_cq_progress_notification` where the enum `ncs_cq_progress_notif_type` type gives the specific event that occurred, see `confd_events.h`. This can be one of `NCS_CQ_ITEM_WAITING` - (waiting on another executing entry), `NCS_CQ_ITEM_EXECUTING`, `NCS_CQ_ITEM_LOCKED` (stalled by parent queue in cluster), `NCS_CQ_ITEM_COMPLETED`, `NCS_CQ_ITEM_FAILED` or `NCS_CQ_ITEM_DELETED`.

CONFD_NOTIF_STREAM_EVENT

This event is generated for a notification stream, i.e. event notifications sent by an application as described in the [NOTIFICATION STREAMS](#) section of [confd_lib_dp\(3\)](#). The application receives a struct `confd_stream_notification` where the enum `confd_stream_notif_type` type gives the specific event that occurred, see `confd_events.h`. This can be either an actual event notification (`CONFD_STREAM_NOTIFICATION_EVENT`), one of `CONFD_STREAM_NOTIFICATION_COMPLETE` or `CONFD_STREAM_REPLAY_COMPLETE`, which indicates that a requested replay has completed, or `CONFD_STREAM_REPLAY_FAILED`, which indicates that a requested replay could not be carried out. In all cases except `CONFD_STREAM_NOTIFICATION_EVENT`, no further `CONFD_NOTIF_STREAM_EVENT` events will be delivered on the socket.

This event also needs to be requested through a call to `confd_notifications_connect2()`, where the required `stream_name` must be provided via the `struct confd_notifications_data` parameter. The additional elements in the struct can be used as follows:

- The `start_time` element can be given to request a replay, in which case `stop_time` can also be given to specify the end of the replay (or "live feed"). The `start_time` and `stop_time` must be set to the type `C_NOEXISTS` to indicate that no value is given, otherwise values of type `C_DATETIME` must be given.
- The `xpath_filter` element may be used to specify an XPath filter to be applied to the notification stream. If no filtering is wanted, `xpath_filter` must be set to `NULL`.
- The `userid` element may be used to specify the id of an existing user session for filtering based on AAA rules. Only notifications that are allowed by the access rights of that user session will be received. If no AAA restrictions are wanted, `userid` must be set to 0.



Note

This event may allocate memory dynamically inside the struct `confd_notification`, thus we must always call `confd_free_notification()` after receiving and processing this event.

Several of the above notification messages contain a lognumber which identifies the event. All log numbers are listed in the file `confd_logsyms.h`. Furthermore the array `confd_log_symbols[]` can be indexed with the lognumber and it contains the symbolic name of each error. The array `confd_log_descriptions[]` can also be indexed with the lognumber and it contains a textual description of the logged event.

FUNCTIONS

The API to receive events from Confd is:

```
int confd_notifications_connect(int sock, const struct sockaddr* srv,
int srv_sz, int mask);

int confd_notifications_connect2(int sock, const struct sockaddr* srv,
int srv_sz, int mask, struct confd_notifications_data *data);
```

These functions create a notification socket. The *mask* is a bitmask of one or several enum *confd_notification_type* values:

```
enum confd_notification_type {
    CONFD_NOTIF_AUDIT                = (1 << 0),
    CONFD_NOTIF_DAEMON               = (1 << 1),
    CONFD_NOTIF_TAKEOVER_SYSLOG      = (1 << 2),
    CONFD_NOTIF_COMMIT_SIMPLE        = (1 << 3),
    CONFD_NOTIF_COMMIT_DIFF          = (1 << 4),
    CONFD_NOTIF_USER_SESSION         = (1 << 5),
    CONFD_NOTIF_HA_INFO              = (1 << 6),
    CONFD_NOTIF_SUBAGENT_INFO        = (1 << 7),
    CONFD_NOTIF_COMMIT_FAILED        = (1 << 8),
    CONFD_NOTIF_SNMPA                = (1 << 9),
    CONFD_NOTIF_FORWARD_INFO         = (1 << 10),
    CONFD_NOTIF_NETCONF              = (1 << 11),
    CONFD_NOTIF_DEVEL                = (1 << 12),
    CONFD_NOTIF_HEARTBEAT            = (1 << 13),
    CONFD_NOTIF_CONFIRMED_COMMIT     = (1 << 14),
    CONFD_NOTIF_UPGRADE_EVENT        = (1 << 15),
    CONFD_NOTIF_COMMIT_PROGRESS      = (1 << 16),
    CONFD_NOTIF_AUDIT_SYNC           = (1 << 17),
    CONFD_NOTIF_HEALTH_CHECK         = (1 << 18),
    CONFD_NOTIF_STREAM_EVENT         = (1 << 19),
    CONFD_NOTIF_HA_INFO_SYNC         = (1 << 20),
    NCS_NOTIF_PACKAGE_RELOAD         = (1 << 21),
    NCS_NOTIF_CQ_PROGRESS            = (1 << 22),
    CONFD_NOTIF_REOPEN_LOGS          = (1 << 23)
};
```

The *confd_notifications_connect2()* variant is required if we wish to subscribe to *CONFD_NOTIF_HEARTBEAT*, *CONFD_NOTIF_HEALTH_CHECK*, or *CONFD_NOTIF_STREAM_EVENT* events. The struct *confd_notifications_data* is defined as:

```
struct confd_notifications_data {
    int heartbeat_interval; /* required if we wish to generate */
                           /* CONFD_NOTIF_HEARTBEAT events */
                           /* the time is milli seconds */
    int health_check_interval; /* required if we wish to generate */
                              /* CONFD_NOTIF_HEALTH_CHECK events */
                              /* the time is milli seconds */
    /* The following five are used for CONFD_NOTIF_STREAM_EVENT */
    char *stream_name; /* stream name (required) */
    confd_value_t start_time; /* type = C_NOEXISTS or C_DATETIME */
    confd_value_t stop_time; /* type = C_NOEXISTS or C_DATETIME */
                           /* when start_time is C_DATETIME */
    char *xpath_filter; /* optional XPath filter for the */
                       /* stream - NULL for no filter */
    int usid; /* optional user session id for */
             /* AAA restriction - 0 for no AAA */
};
```

When requesting the `CONFID_NOTIF_STREAM_EVENT` event, `confd_notifications_connect2()` may fail and return `CONFID_ERR`, with some specific `confd_errno` values:

<code>CONFID_ERR_NOEXISTS</code>	The stream name given by <code>stream_name</code> does not exist.
<code>CONFID_ERR_XPATH</code>	The XPath filter provided via <code>xpath_filter</code> failed to compile.
<code>CONFID_ERR_NOSESSION</code>	The user session id given by <code>usid</code> does not identify an existing user session.



Note

If these calls fail (i.e. do not return `CONFID_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

```
int confd_read_notification(int sock, struct confd_notification *n);
```

The application is responsible for polling the notification socket. Once data is available to be read on the socket the application must call `confd_read_notification()` to read the data from the socket. On success the function returns `CONFID_OK` and populates the `struct confd_notification*` pointer. See `confd_events.h` for the definition of the `struct confd_notification` structure.

If the application is not reading from the socket and a `write()` from ConfD hangs for more than 15 seconds, ConfD will close the socket and log the event to the `confdLog`.

```
void confd_free_notification(struct confd_notification *n);
```

The `struct confd_notification` can sometimes have memory dynamically allocated inside it. Currently the notification types that render structures with allocated memory inside them are `CONFID_NOTIF_SNMPA` and `CONFID_NOTIF_STREAM_EVENT`. If such an event is received, this function must be called to free any memory allocated inside the received notification structure.

For those notification structures that do not have any memory allocated, this function is a no-op, thus it is always safe to call this function after a notification structure has been processed.

```
int confd_diff_notification_done(int sock, struct confd_trans_ctx *tctx);
```

If the received event was `CONFID_NOTIF_COMMIT_DIFF` it is important that we call this function when we are done reading the transaction diffs over MAAPI. The transaction is hanging until this function gets called. This function also releases memory associated to the transaction in the library.

```
int confd_sync_audit_notification(int sock, int usid);
```

If the received event was `CONFID_NOTIF_AUDIT`, and we are subscribing to notifications with the flag `CONFID_NOTIF_AUDIT_SYNC`, this function must be called when we are done processing the notification. The user session is hanging until this function gets called.

```
int confd_sync_ha_notification(int sock);
```

If the received event was `CONFID_NOTIF_HA_INFO`, and we are subscribing to notifications with the flag `CONFID_NOTIF_HA_INFO_SYNC`, this function must be called when we are done processing the notification. All HA processing is blocked until this function gets called.

SEE ALSO

The ConfD User Guide

Name

confd_lib_ha — library for connecting to NSO HA subsystem

Synopsis

```
#include <confd_lib.h> #include <confd_ha.h>

int confd_ha_connect(int sock, const struct sockaddr* srv, int srv_sz,
const char *token);

int confd_ha_bemaster(int sock, confd_value_t *mynodeid);

int confd_ha_beslave(int sock, confd_value_t *mynodeid, struct
confd_ha_node *master, int waitreply);

int confd_ha_berelay(int sock);

int confd_ha_benone(int sock);

int confd_ha_status(int sock, struct confd_ha_status *stat);

int confd_ha_slave_dead(int sock, confd_value_t *nodeid);
```

LIBRARY

ConfD Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to the NSO High Availability (HA) subsystem. NSO can replicate the configuration data on several nodes in a cluster. The purpose of this API is to manage the HA functionality. The details on usage of the HA API are described in the chapter [High availability](#) in the User Guide.

FUNCTIONS

```
int confd_ha_connect(int sock, const struct sockaddr* srv, int srv_sz,
const char *token);
```

Connect a HA socket which can be used to control a NSO HA node. The token is a secret string that must be shared by all participants in the cluster. There can only be one HA socket towards NSO, a new call to `confd_ha_connect()` makes NSO close the previous connection and reset the token to the new value. Returns CONFD_OK or CONFD_ERR.



Note

If this call fails (i.e. does not return CONFD_OK), the socket descriptor must be closed and a new socket created before the call is re-attempted.

```
int confd_ha_bemaster(int sock, confd_value_t *mynodeid);
```

Instruct a HA node to be master and also give the node a name. Returns CONFD_OK or CONFD_ERR.

Errors: CONFD_ERR_HA_BIND if we cannot bind the TCP socket, CONFD_ERR_BADSTATE if NSO is still in start phase 0.

```
int confd_ha_beslave(int sock, confd_value_t *mynodeid, struct
confd_ha_node *master, int waitreply);
```

Instruct a NSO HA node to be slave with a named master. The *waitreply* is a boolean int. If 1, the function is synchronous and it will hang until the node has initialized its CDB database. This may mean that the CDB database is copied in its entirety from the master. If 0, we do not wait for the reply, but it is possible to use a notifications socket and get notified asynchronously via a HA_INFO_BESLAVE_RESULT notification. In both cases, it is also possible to use a notifications socket and get notified asynchronously when CDB at the slave is initialized.

If the call of this function fails with *confd_errno* CONFD_ERR_HA_CLOSED, it means that the initial synchronization with the master failed, either due to the socket being closed or due to a timeout while waiting for a response from the master. The function will fail with error CONFD_ERR_BADSTATE if NSO is still in start phase 0.

Errors: CONFD_ERR_HA_CONNECT, CONFD_ERR_HA_BADNAME, CONFD_ERR_HA_BADTOKEN, CONFD_ERR_HA_BADFXS, CONFD_ERR_HA_BADVSN, CONFD_ERR_HA_CLOSED, CONFD_ERR_BADSTATE

```
int confd_ha_berelay(int sock);
```

Instruct an established HA slave node to be a relay for other slaves. This can be useful in certain deployment scenarios, but makes the management of the cluster more complex. Read more about this in the [Relay slaves](#) section of the High availability chapter in the User Guide. Returns CONFD_OK or CONFD_ERR.

Errors: CONFD_ERR_HA_BIND if we cannot bind the TCP socket, CONFD_ERR_BADSTATE if the node is not already a slave.

```
int confd_ha_benone(int sock);
```

Instruct a node to resume the initial state, i.e. neither master nor slave.

Errors: CONFD_ERR_BADSTATE if NSO is still in start phase 0.

```
int confd_ha_status(int sock, struct confd_ha_status *stat);
```

Query a NSO HA node for its status. If successful, the function populates the *confd_ha_status* structure. This is the only HA related function which is possible to call while the NSO daemon is still in start phase 0.

```
int confd_ha_slave_dead(int sock, confd_value_t *nodeid);
```

This function must be used by the application to inform NSO HA subsystem that another node which is possibly connected to NSO is dead.

Errors: CONFD_ERR_BADSTATE if NSO is still in start phase 0.

SEE ALSO

`confd.conf(5)` - ConfD daemon configuration file format

The NSO User Guide

Name

confd_lib_lib — common library functions for applications connecting to NSO

Synopsis

```
#include <confd_lib.h>

void confd_init(const char *name, FILE *estream, const enum
confd_debug_level debug);

int confd_set_debug(enum confd_debug_level debug, FILE *estream);

void confd_fatal(const char *fmt, ...);

int confd_load_schemas(const struct sockaddr* srv, int srv_sz);

int confd_load_schemas_list(const struct sockaddr* srv, int srv_sz, int
flags, const u_int32_t *nshash, const int *nsflags, int num_ns);

int confd_mmap_schemas_setup(void *addr, size_t size, const char
*filename, int flags);

int confd_mmap_schemas(const char *filename);

void confd_free_schemas(void);

int confd_svcmp(const char *s, const confd_value_t *v);

int confd_pp_value(char *buf, int bufsiz, const confd_value_t *v);

int confd_ns_pp_value(char *buf, int bufsiz, const confd_value_t *v,
int ns);

int confd_pp_kpath(char *buf, int bufsiz, const confd_hkeypath_t
*hkeypath);

int confd_pp_kpath_len(char *buf, int bufsiz, const confd_hkeypath_t
*hkeypath, int len);

char *confd_xmltag2str(u_int32_t ns, u_int32_t xmltag);

int confd_xpath_pp_kpath(char *buf, int bufsiz, u_int32_t ns, const
confd_hkeypath_t *hkeypath);

int confd_format_keypath(char *buf, int bufsiz, const char *fmt, ...);

int confd_vformat_keypath(char *buf, int bufsiz, const char *fmt,
va_list ap);

int confd_get_nslist(struct confd_nsinfo **listp);

char *confd_ns2prefix(u_int32_t ns);
```

```

char *confd_hash2str(u_int32_t hash);

u_int32_t confd_str2hash(const char *str);

struct confd_cs_node *confd_find_cs_root(int ns);

struct confd_cs_node *confd_find_cs_node(const confd_hkeypath_t
*hkeypath, int len);

struct confd_cs_node *confd_find_cs_node_child(const struct
confd_cs_node *parent, struct xml_tag xmltag);

struct confd_cs_node *confd_cs_node_cd(const struct confd_cs_node
*start, const char *fmt, ...);

int confd_max_object_size(struct confd_cs_node *object);

struct confd_cs_node *confd_next_object_node(struct confd_cs_node
*object, struct confd_cs_node *cur, confd_value_t *value);

struct confd_type *confd_find_ns_type(u_int32_t nshash, const char
*name);

struct confd_type *confd_get_leaf_list_type(struct confd_cs_node
*node);

int confd_val2str(struct confd_type *type, const confd_value_t *val,
char *buf, int bufsiz);

int confd_str2val(struct confd_type *type, const char *str,
confd_value_t *val);

char *confd_val2str_ptr(struct confd_type *type, const confd_value_t
*val);

int confd_get_decimal64_fraction_digits(struct confd_type *type);

int confd_get_bitbig_size(struct confd_type *type);

int confd_hkp_tagmatch(struct xml_tag tags[], int tagslen,
confd_hkeypath_t *hkp);

int confd_hkp_prefix_tagmatch(struct xml_tag tags[], int tagslen,
confd_hkeypath_t *hkp);

int confd_val_eq(const confd_value_t *v1, const confd_value_t *v2);

void confd_free_value(confd_value_t *v);

confd_value_t *confd_value_dup_to(const confd_value_t *v, confd_value_t
*newv);

void confd_free_dup_to_value(confd_value_t *v);

confd_value_t *confd_value_dup(const confd_value_t *v);

```

```

void confd_free_dup_value(confd_value_t *v);

confd_hkeypath_t *confd_hkeypath_dup(const confd_hkeypath_t *src);

confd_hkeypath_t *confd_hkeypath_dup_len(const confd_hkeypath_t *src,
int len);

void confd_free_hkeypath(confd_hkeypath_t *hkp);

void confd_free_authorization_info(struct confd_authorization_info
*ainfo);

char *confd_lasterr(void);

char *confd_strerror(int code);

struct xml_tag *confd_last_error_apptag(void);

int confd_register_ns_type(u_int32_t nshash, const char *name, struct
confd_type *type);

int confd_register_node_type(struct confd_cs_node *node, struct
confd_type *type);

int confd_type_cb_init(struct confd_type_cbs **cbs);

int confd_decrypt(const char *ciphertext, int len, char *output);

int confd_stream_connect(int sock, const struct sockaddr* srv, int
srv_sz, int id, int flags);

int confd_deserialize(struct confd_deserializable *s, unsigned char
*buf);

int confd_serialize(struct confd_serializable *s, unsigned char *buf,
int buf_sz, int *bytes_written, unsigned char **allocated);

void confd_deserialized_free(struct confd_deserializable *s);

```

LIBRARY

NSO Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to NSO. This manual page describes functions and data structures that are not specific to any one of the APIs that are described in the other confd_lib_xxx(3) manual pages.

FUNCTIONS

```

void confd_init(const char *name, FILE *estream, const enum
confd_debug_level debug);

```

Initializes the ConfD library. Must be called before any other NSO API functions are called.

The *debug* parameter is used to control the debug level. The following levels are available:

CONF_D_SILENT	No printouts whatsoever are produced by the library.
CONF_D_DEBUG	Various printouts will occur for various error conditions. This is a decent value to have as default. If syslog is enabled for the library, these printouts will be logged at syslog level LOG_ERR, except for errors where <code>confd_errno</code> is CONF_D_ERR_INTERNAL, which are logged at syslog level LOG_CRIT.
CONF_D_TRACE	The execution of callback functions and CDB/MAAPI API calls will be traced. This is very verbose and very useful during debugging. If syslog is enabled for the library, these printouts will be logged at syslog level LOG_DEBUG.
CONF_D_PROTO_TRACE	The low-level protocol exchange between the application and NSO will be traced. This is even more verbose than CONF_D_TRACE, and normally only of interest to Tail-f support. These printouts will not be logged via syslog, i.e. a non-NULL value for the <i>estream</i> parameter must be provided.

The *estream* parameter is used by all printouts from the library. The *name* parameter is typically included in most of the debug printouts. If the *estream* parameter is NULL, no printouts to a file will occur. Independent of the *estream* parameter, syslog can be enabled for the library by setting the global variable `confd_lib_use_syslog` to 1. See [SYSLOG AND DEBUG](#) in this man page.

```
int confd_set_debug(enum confd_debug_level debug, FILE *estream);
```

This function can be used to change the *estream* and *debug* parameters for the library.

```
int confd_load_schemas(const struct sockaddr* srv, int srv_sz);
```

Utility function that uses `maapi_load_schemas()` (see [confd_lib_maapi\(3\)](#)) to load schema information from NSO. This function connects to NSO and loads all the schema information in NSO for all loaded "fxs" files into the library. This is necessary in order to get proper printouts of e.g. `confd_hkeypaths` which otherwise just contains arrays of integers. This function should typically always be called when we initialize the library. See [confd_types\(3\)](#).

```
int confd_load_schemas_list(const struct sockaddr* srv, int srv_sz, int flags, const u_int32_t *nshash, const int *nsflags, int num_ns);
```

Utility function that uses `maapi_load_schemas_list()` to load a subset of the schema information from NSO. See the description of `maapi_load_schemas_list()` in [confd_lib_maapi\(3\)](#) for the details of how to use the *flags*, *nshash*, *nsflags*, and *num_ns* parameters.

```
int confd_mmap_schemas_setup(void *addr, size_t size, const char *filename, int flags);
```

This function sets up for a subsequent call of one of the schema-loading functions (`confd_load_schemas()` etc) to load the schema information into a shared memory segment instead of into the process' heap. See the section [Using shared memory for schema information](#) in the Advanced Topics chapter in the User Guide for usage discussion. The *addr* and (potentially) *size* arguments are passed to `mmap(2)`, and *filename* specifies the pathname of a file to use as backing store. The *flags* parameter can be given as CONF_D_MMAP_SCHEMAS_KEEP_SIZE to request that the shared memory

segment should be exactly the size given by the (non-zero) *size* argument - if this size is insufficient to hold the schema information, the schema-loading function will fail.

```
int confd_mmap_schemas(const char *filename);
```

Map a shared memory segment, previously created by `confd_mmap_schemas_setup()` and subsequent schema loading, into the current process' address space, and make it ready for use. The *filename* argument specifies the pathname of the file that is used as backing store. See also `/ncs-config/enable-shared-memory-schema` in [ncs.conf\(5\)](#) and `maapi_get_schema_file_path()` in [confd_lib_maapi\(3\)](#).

```
void confd_free_schemas(void);
```

Free or unmap the memory allocated or mapped by schema loading, undoing the result of loading - i.e. schema information will no longer be available. There is normally no need to call this function, since the memory will be automatically freed/unmapped if a new schema loading is done, or when the process terminates, but it may be useful in some cases.

```
int confd_svcmp(const char *s, const confd_value_t *v);
```

Utility function with similar semantics to `strcmp()` which compares a `confd_value_t` to a `char*`.

```
int confd_pp_value(char *buf, int bufsiz, const confd_value_t *v);
```

Utility function which pretty prints up to *bufsiz* characters into *buf*, giving a string representation of the value *v*. Since only the "primitive" type as defined by the enum `confd_vtype` is available, `confd_pp_value()` can not produce a true string representation in all cases, see the list below. If this is a problem, use `confd_val2str()` instead.

<code>C_ENUM_VALUE</code>	The value is printed as "enum<N>", where N is the integer value.
<code>C_BIT32</code>	The value is printed as "bits<X>", where X is an unsigned integer in hexadecimal format.
<code>C_BIT64</code>	The value is printed as "bits<X>", where X is an unsigned integer in hexadecimal format.
<code>C_BITBIG</code>	The value is printed as "bits<X>", where X is an unsigned integer (possibly very large) in hexadecimal format.
<code>C_BINARY</code>	The string representation for <code>xs:hexBinary</code> is used, i.e. a sequence of hexadecimal characters.
<code>C_DECIMAL64</code>	If the value of the <code>fraction_digits</code> element is within the possible range (1..18), it is assumed to be correct for the type and used for the string representation. Otherwise the value is printed as "invalid64<N>", where N is the value of the value element.
<code>C_XMLTAG</code>	The string representation is printed if schema information has been loaded into the library. Otherwise the value is printed as "tag<N>", where N is the integer value.
<code>C_IDENTITYREF</code>	The string representation is printed if schema information has been loaded into the library. Otherwise the value is printed as "idref<N>", where N is the integer value.

All the pp pretty print functions, i.e. `confd_pp_value()` `confd_ns_pp_value()`, `confd_pp_kpath()` and `confd_xpath_pp_kpath()`, as well as the `confd_format_keypath()` and `confd_val2str()` functions, return the number of characters printed (not including the trailing NUL used to end output to strings) if there is enough space.

The formatting functions do not write more than *bufsiz* bytes (including the trailing NUL). If the output was truncated due to this limit then the return value is the number of characters (not including the trailing NUL) which would have been written to the final string if enough space had been available. Thus, a return value of *bufsiz* or more means that the output was truncated.

Except for `confd_val2str()`, these functions will never return `CONF_ERR` or any other negative value.

```
int confd_ns_pp_value(char *buf, int bufsiz, const confd_value_t *v,
int ns);
```

This function is deprecated, but will remain for backward compatibility. It just calls `confd_pp_value()` - use `confd_pp_value()` directly, or `confd_val2str()` (see below), instead.

```
int confd_pp_kpath(char *buf, int bufsiz, const confd_hkeypath_t
*hkeypath);
```

Utility function which pretty prints up to *bufsiz* characters into *buf*, giving a string representation of the path *hkeypath*. This will use the NSO curly brace notation, i.e. `/servers/server{www}/ip`. Requires that schema information is available to the library, see [confd_types\(3\)](#). Same return value as `confd_pp_value()`.

```
int confd_pp_kpath_len(char *buf, int bufsiz, const confd_hkeypath_t
*hkeypath, int len);
```

A variant of `confd_pp_kpath()` that prints only the first *len* elements of *hkeypath*.

```
int confd_format_keypath(char *buf, int bufsiz, const char *fmt, ...);
```

Several of the functions in [confd_lib_maapi\(3\)](#) and [confd_lib_cdb\(3\)](#) take a variable number of arguments which are then, similar to `printf`, used to generate the path passed to NSO - see the [PATHS](#) section of [confd_lib_cdb\(3\)](#). This function takes the same arguments, but only formats the path as a string, writing at most *bufsiz* characters into *buf*. If the path is absolute and schema information is available to the library, key values referenced by a `"%x"` modifier will be printed according to their specific type, i.e. effectively using `confd_val2str()`, otherwise `confd_pp_value()` is used. Same return value as `confd_pp_value()`.

```
int confd_vformat_keypath(char *buf, int bufsiz, const char *fmt,
va_list ap);
```

Does the same as `confd_format_keypath()`, but takes a single *va_list* argument instead of a variable number of arguments - i.e. similar to `vprintf`. Same return value as `confd_pp_value()`.

```
char *confd_xmltag2str(u_int32_t ns, u_int32_t xmltag);
```

This function is deprecated, but will remain for backward compatibility. It just calls `confd_hash2str()` - use `confd_hash2str()` directly instead, see below.

```
int confd_xpath_pp_kpath(char *buf, int bufsiz, u_int32_t ns, const
confd_hkeypath_t *hkeypath);
```


Similar to `confd_pp_kpath()` except that the path is formatted as an XPath path, i.e. `"/servers:servers/server[name="www"]/ip"`. This function can also take the namespace integer as an argument. If 0 is passed as *ns*, the namespace is derived from the *hkeypath*. Requires that schema information is available to the library, see [confd_types\(3\)](#). Same return value as `confd_pp_value()`.

```
int confd_get_nslist(struct confd_nsinfo **listp);
```

Provides a list of the namespaces known to the library as an array of struct `confd_nsinfo` structures:

```
struct confd_nsinfo {  
    const char *uri;  
    const char *prefix;  
    u_int32_t hash;  
    const char *revision;  
};
```

A pointer to the array is stored in **listp*, and the function returns the number of elements in the array. The *revision* element in struct `confd_nsinfo` will give the revision for YANG modules that have a revision statement, otherwise it is NULL.

```
char *confd_ns2prefix(u_int32_t ns);
```

Returns a NUL-terminated string giving the namespace prefix for the namespace *ns*, if the namespace is known to the library - otherwise it returns NULL.

```
char *confd_hash2str(u_int32_t hash);
```

Returns a NUL-terminated string representing the node name given by *hash*, or NULL if the hash value is not found. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#) - otherwise it always returns NULL.

```
u_int32_t confd_str2hash(const char *str);
```

Returns the hash value representing the node name given by *str*, or 0 if the string is not found. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#) - otherwise it always returns 0.

```
struct confd_cs_node *confd_find_cs_root(int ns);
```

When schema information is available to the library, this function returns the root of the tree representation of the namespace given by *ns*, i.e. a pointer to the struct `confd_cs_node` for the (first) toplevel node. For namespaces that are augmented into other namespaces such that they do not have a toplevel node, this function returns NULL - the nodes of such a namespace are found below the augment target node(s) in other tree(s). See [confd_types\(3\)](#).

```
struct confd_cs_node *confd_find_cs_node(const confd_hkeypath_t  
*hkeypath, int len);
```

Utility function which finds the struct `confd_cs_node` corresponding to the *len* first elements of the hashed keypath. To make the search consider the full keypath, pass the *len* element from the `confd_hkeypath_t` structure (i.e. `mykeypath->len`). See [confd_types\(3\)](#).

```
struct confd_cs_node *confd_find_cs_node_child(const struct
confd_cs_node *parent, struct xml_tag xmltag);
```

Utility function which finds the struct `confd_cs_node` corresponding to the child node given as `xmltag`. See [confd_types\(3\)](#).

```
struct confd_cs_node *confd_cs_node_cd(const struct confd_cs_node
*start, const char *fmt, ...);
```

Utility function which finds the resulting struct `confd_cs_node` given an (optional) starting node and a (relative or absolute) string keypath. I.e. this function navigates the tree in a manner corresponding to `cdb_cd()`/`maapi_cd()`. Note however that the `confd_cs_node` tree does not have a node corresponding to `"/"`. It is possible to pass `start` as `NULL`, in which case the path must be absolute (i.e. start with a `"/"`).

Since the key values are not relevant for the tree navigation, the key elements can be omitted, i.e. a "tagpath" can be used - if present, key elements are ignored, whether given in the `{...}` form or the CDB-only `[N]` form. See [confd_types\(3\)](#).

If the path can not be found, `NULL` is returned, `confd_errno` is set to `CONFID_ERR_BADPATH`, and `confd_lasterr()` can be used to retrieve a string that describes the reason for the failure.

```
int confd_max_object_size(struct confd_cs_node *object);
```

Utility function which returns the maximum size (i.e. the needed length of the `confd_value_t` array) for an "object" retrieved by `cdb_get_object()`, `maapi_get_object()`, and corresponding multi-object functions. The `object` parameter is a pointer to the list or container `confd_cs_node` node for which we want to find the maximum size. See the description of `cdb_get_object()` in [confd_lib_cdb\(3\)](#) for usage examples.

```
struct confd_cs_node *confd_next_object_node(struct confd_cs_node
*object, struct confd_cs_node *cur, confd_value_t *value);
```

Utility function to allow navigation of the `confd_cs_node` schema tree in parallel with the `confd_value_t` array populated by `cdb_get_object()`, `maapi_get_object()`, and corresponding multi-object functions. The `object` parameter is a pointer to the list or container node as for `confd_max_object_size()`, the `cur` parameter is a pointer to the `confd_cs_node` node for the current value, and the `value` parameter is a pointer to the current value in the array. The function returns a pointer to the `confd_cs_node` node for the next value in the array, or `NULL` when the complete object has been traversed. In the initial call for a given traversal, we must pass `object->children` for the `cur` parameter - this always points to the `confd_cs_node` node for the first value in the array. See the description of `cdb_get_object()` in [confd_lib_cdb\(3\)](#) for usage examples.

```
struct confd_type *confd_find_ns_type(u_int32_t nshash, const char
*name);
```

Returns a pointer to a type definition for the type named `name`, which is defined in the namespace identified by `nshash`, or `NULL` if the type could not be found. If `nshash` is 0, the type name will be looked up among the ConfD built-in types (i.e. the YANG built-in types, the types defined in the YANG "tailf-common" module, and the types defined in the "confd" and "xs" namespaces). The type definition pointer can be used with the `confd_val2str()` and `confd_str2val()` functions, see below. If

nshash is not 0, the function requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#) - otherwise it returns NULL.

```
struct confd_type *confd_get_leaf_list_type(struct confd_cs_node
*node);
```

For a leaf-list node, the *type* field in the struct *confd_cs_node_info* (see [confd_types\(3\)](#)) identifies a "list type" for the leaf-list "itself". This function takes a pointer to the struct *confd_cs_node* for a leaf-list node as argument, and returns the type of the elements in the leaf-list, i.e. corresponding to the *type* substatement for the leaf-list in the YANG module. If called for a node that is not a leaf-list, it returns NULL and sets *confd_errno* to *CONFID_ERR_PROTOUSAGE*. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#) - otherwise it returns NULL and sets *confd_errno* to *CONFID_ERR_UNAVAILABLE*.

```
int confd_val2str(struct confd_type *type, const confd_value_t *val,
char *buf, int bufsiz);
```

Prints the string representation of *val* into *buf*, which has the length *bufsiz*, using type information from the data model. Returns the length of the string as described for *confd_pp_value()*, or *CONFID_ERR* if the value could not be converted (e.g. wrong type). The *type* pointer can be obtained either from the struct *confd_cs_node* corresponding to the leaf that *val* pertains to, or via the *confd_find_ns_type()* function above. The struct *confd_cs_node* can in turn be obtained by various combinations of the functions that operate on the *confd_cs_node* trees (see above), or by user-defined functions for navigating those trees. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#).

```
int confd_str2val(struct confd_type *type, const char *str,
confd_value_t *val);
```

Stores the value corresponding to the NUL-terminated string *str* in *val*, using type information from the data model. Returns *CONFID_OK*, or *CONFID_ERR* if the string could not be converted. See *confd_val2str()* for a description of the *type* argument. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#).



Note

When the resulting value is of one of the *C_BUF*, *C_BINARY*, *C_LIST*, *C_OBJECTREF*, *C_OID*, *C_QNAME*, *C_HEXSTR*, or *C_BITBIG* *confd_value_t* types, the library has allocated memory to hold the value. It is up to the user of this function to free the memory using *confd_free_value()*.

```
char *confd_val2str_ptr(struct confd_type *type, const confd_value_t
*val);
```

A variant of *confd_val2str()* that can be used only when the string representation is a constant, i.e. *C_ENUM_VALUE* values. In this case it returns a pointer to the string, otherwise NULL. See *confd_val2str()* for a description of the *type* argument. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#).

```
int confd_get_decimal64_fraction_digits(struct confd_type *type);
```

Utility function to obtain the value of the argument to the *fraction-digits* statement for a YANG decimal64 type. This is useful when we want to create a *confd_value_t* for such a type, since the value

element must be scaled according to the fraction-digits value. The function returns the fraction-digits value, or 0 if the *type* argument does not refer to a decimal64 type. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#).

```
int confd_get_bitbig_size(struct confd_type *type);
```

Utility function to obtain the maximum size needed for the byte array for the C_BITBIG *confd_value_t* representation used when a YANG bits type has a highest bit position above 63. This is useful when we want to create a *confd_value_t* for such a type, since an array of this size can hold the values for all the bits defined for the type. Applications may however provide a *confd_value_t* with a shorter (but not longer) array to NSO. The file generated by **ncsc --emit-h** also includes a `#define` symbol for this size. The function returns 0 if the *type* argument does not refer to a bits type with a highest bit position above 63. Requires that schema information has been loaded from the NSO daemon into the library, see [confd_types\(3\)](#).

```
int confd_hkp_tagmatch(struct xml_tag tags[], int tagslen,  
confd_hkeypath_t *hkp);
```

When checking the hkeypaths that get passed into each iteration in e.g. *cdb_diff_iterate()* we can either explicitly check the paths, or use this function to do the job. The *tags* array (typically statically initialized) specifies a tagpath to match against the hkeypath. See *cdb_diff_match()*. The function returns one of these values:

```
#define CONFID_HKP_MATCH_NONE 0  
#define CONFID_HKP_MATCH_TAGS (1 << 0)  
#define CONFID_HKP_MATCH_HKP (1 << 1)  
#define CONFID_HKP_MATCH_FULL (CONFID_HKP_MATCH_TAGS | CONFID_HKP_MATCH_HKP)
```

CONFID_HKP_MATCH_TAGS means that the whole tagpath was matched by the hkeypath, and CONFID_HKP_MATCH_HKP means that the whole hkeypath was matched by the tagpath.

```
int confd_hkp_prefix_tagmatch(struct xml_tag tags[], int tagslen,  
confd_hkeypath_t *hkp);
```

A simplified version of *confd_hkp_tagmatch()* - it returns 1 if the tagpath matches a prefix of the hkeypath, i.e. it is equivalent to calling *confd_hkp_tagmatch()* and checking if the return value includes CONFID_HKP_MATCH_TAGS.

```
int confd_val_eq(const confd_value_t *v1, const confd_value_t *v2);
```

Utility function which compares two values. Returns positive value if equal, 0 otherwise.

```
void confd_fatal(const char *fmt, ...);
```

Utility function which formats a string, prints it to stderr and exits with exit code 1.

```
void confd_free_value(confd_value_t *v);
```

When we retrieve values via the CDB or MAAPI interfaces, or convert strings to values via *confd_str2val()*, and these values are of either of the types C_BUF, C_BINARY, C_QNAME, C_OBJECTREF, C_OID, C_LIST, C_HEXSTR, or C_BITBIG, the library has allocated memory to hold the values. This memory must be freed by the application when it is done with the value. This function frees memory for all *confd_value_t* types. Note that this function does not free the structure

itself, only possible internal pointers inside the struct. Typically we use `confd_value_t` variables as automatic variables allocated on the stack. If the held value is of fixed size, e.g. integers, xmltags etc, the `confd_free_value()` function does nothing.



Note

Memory for values received as parameters to callback functions is always managed by the library - the application must *not* call `confd_free_value()` for those (on the other hand values of the types listed above that are received as parameters to a callback function must be copied if they are to persist beyond the callback invocation).

```
confd_value_t *confd_value_dup_to(const confd_value_t *v, confd_value_t
*newv);
```

This function copies the contents of `*v` to `*newv`, allocating memory for the actual value for the types that need it. It returns `newv`, or NULL if allocation failed. The allocated memory (if any) can be freed with `confd_free_dup_to_value()`.

```
void confd_free_dup_to_value(confd_value_t *v);
```

Frees memory allocated by `confd_value_dup_to()`. Note this is not the same as `confd_free_value()`, since `confd_value_dup_to()` also allocates memory for values of type `C_STR` - such values are not freed by `confd_free_value()`.

```
confd_value_t *confd_value_dup(const confd_value_t *v);
```

This function allocates memory and duplicates `*v`, i.e. a `confd_value_t` struct is always allocated, memory for the actual value is also allocated for the types that need it. Returns a pointer to the new `confd_value_t`, or NULL if allocation failed. The allocated memory can be freed with `confd_free_dup_value()`.

```
void confd_free_dup_value(confd_value_t *v);
```

Frees memory allocated by `confd_value_dup()`. Note this is not the same as `confd_free_value()`, since `confd_value_dup()` also allocates the actual `confd_value_t` struct, and allocates memory for values of type `C_STR` - such values are not freed by `confd_free_value()`.

```
confd_hkeypath_t *confd_hkeypath_dup(const confd_hkeypath_t *src);
```

This function allocates memory and duplicates a `confd_hkeypath_t`.

```
confd_hkeypath_t *confd_hkeypath_dup_len(const confd_hkeypath_t *src,
int len);
```

Like `confd_hkeypath_dup()`, but duplicates only the first `len` elements of the `confd_hkeypath_t`. I.e. the elements are shifted such that `v[0][0]` still refers to the last element.

```
void confd_free_hkeypath(confd_hkeypath_t *hkp);
```

This function will free memory allocated by e.g. `confd_hkeypath_dup()`.

```
void confd_free_authorization_info(struct confd_authorization_info
*ainfo);
```

This function will free memory allocated by `maapi_get_authorization_info()`.

```
int confd_decrypt(const char *ciphertext, int len, char *output);
```

When data is read over the CDB interface, the MAAPI interface or received in event notifications, the data for the two builtin types `tailf:des3-cbc-encrypted-string` or `tailf:aes-cfb-128-encrypted-string` is encrypted.

This function decrypts `len` bytes of data from `ciphertext` and writes the clear text to the `output` pointer. The `output` pointer must point to an area that is at least `len` bytes long.



Note

One of the functions `confd_install_crypto_keys()` and `maapi_install_crypto_keys()` must have been called before `confd_decrypt()` can be used.

USER-DEFINED TYPES

It is possible to define new types, i.e. mappings between a textual representation and a `confd_value_t` representation that are not pre-defined in the NSO daemon. Read more about this in the [confd_types\(3\)](#) manual page.

```
int confd_type_cb_init(struct confd_type_cbs **cbs);
```

This is the prototype for the function that a shared object implementing one or more user-defined types must provide. See [confd_types\(3\)](#).

```
int confd_register_ns_type(u_int32_t nshash, const char *name, struct confd_type *type);
```

This function can be used to register a user-defined type with the libconfd library, to make it possible for `confd_str2val()` and `confd_val2str()` to provide local string<->value translation in the application. See [confd_types\(3\)](#).

```
int confd_register_node_type(struct confd_cs_node *node, struct confd_type *type);
```

This function provides an alternate way to register a user-defined type with the libconfd library, in particular when the user-defined type is specified "inline" in a `leaf` or `leaf-list` statement. See [confd_types\(3\)](#).

CONF D STREAMS

Some functions in the NSO lib stream data. Either from NSO to the application or from the application to NSO. The individual functions that use this feature will explicitly indicate that the data is passed over a stream socket.

```
int confd_stream_connect(int sock, const struct sockaddr* srv, int srv_sz, int id, int flags);
```

Connects a stream socket to NSO. The `id` and the `flags` take different values depending on the usage scenario. This is indicated for each individual function that makes use of a stream socket.

**Note**

If this call fails (i.e. does not return `CONFID_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

MARSHALLING

In various distributed scenarios we may want to send `confd_lib` datatypes over the network. We have support to marshal and unmarshal some key datatypes.

```
int confd_serialize(struct confd_serializable *s, unsigned char *buf,
int bufsz, int *bytes_written, unsigned char **allocated);
```

This function takes a `confd_serializable` struct as parameter. We have:

```
enum confd_serializable_type {
    CONFID_SERIAL_NONE      = 0,
    CONFID_SERIAL_VALUE_T   = 1,
    CONFID_SERIAL_HKEYPATH  = 2,
    CONFID_SERIAL_TAG_VALUE = 3
};

struct confd_serializable {
    enum confd_serializable_type type;
    union {
        confd_value_t *value;
        confd_hkeypath_t *hkp;
        confd_tag_value_t *tval;
    } u;
};
```

The structure must be populated with a valid type and also a value to be serialized. The serialized data will be written into the provided buffer. If the size of the buffer is insufficient, the function returns the required size as a positive integer. If the provided buffer is `NULL`, the function will allocate a buffer and it is the responsibility of the caller to free the buffer. The optionally allocated buffer is then returned in the output char ** parameter `allocated`. The function returns 0 on success and -1 on failures.

```
int confd_deserialize(struct confd_deserializable *s, unsigned char
*buf);
```

This function takes a `confd_deserializable` struct as parameter. We have:

```
struct confd_deserializable {
    enum confd_serializable_type type;
    union {
        confd_value_t value;
        confd_hkeypath_t hkp;
        confd_tag_value_t tval;
    } u;
    void *internal; // internal structure containing memory
                  // for the above datatypes to point _into_
                  // freed by a call to confd_deserialize_free()
};
```

This function is the reverse of `confd_serialize()`. It populates the provided `confd_deserializable` structure with a type indicator and a reproduced value of the correct type. The structure contains allocated memory that must subsequently be freed with `confd_deserialize_free()`.

```
void confd_deserialized_free(struct confd_deserializable *s);
```

A populated `confd_deserializable` struct contains allocated memory that must be freed. This function traverses a `confd_deserializable` struct as populated by the `confd_deserialize()` function and frees all allocated memory.

EXTENDED ERROR REPORTING

The data provider callback functions described in [confd_lib_dp\(3\)](#) can pass error information back to NSO either as a simple string using `confd_xxx_seterr()`, or in a more structured/detailed form using the corresponding `confd_xxx_seterr_extended()` function. This form is also used when a CDB subscriber wishes to abort the current transaction with `cdb_sub_abort_trans()`, see [confd_lib_cdb\(3\)](#). There is also a set of `confd_xxx_seterr_extended_info()` functions and a `cdb_sub_abort_trans_info()` function, that can alternatively be used if we want to provide contents for the NETCONF <error-info> element. The description below uses the functions for transaction callbacks as an example, but the other functions follow the same pattern:

```
void confd_trans_seterr_extended(struct confd_trans_ctx *tctx, enum
confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag, const
char *fmt, ...);
```

The function can be used also after a data provider callback has returned `CONF_DELAYED_RESPONSE`, but in that case it must be followed by a call of `confd_delayed_reply_error()` (see [confd_lib_dp\(3\)](#)) with `NULL` for the `errstr` pointer.

One of the following values can be given for the `code` argument:

`CONF_ERRCODE_IN_USE`

Locking a data store was not possible because it was already locked.

`CONF_ERRCODE_RESOURCE_DENIED`

General resource unavailability, e.g. insufficient memory to carry out an operation.

`CONF_ERRCODE_INCONSISTENT_VALUE`

A request parameter had an unacceptable/invalid value

`CONF_ERRCODE_ACCESS_DENIED`

The request could not be fulfilled because authorization did not allow it. (No additional error information will be reported by the northbound agent, to avoid any security breach.)

`CONF_ERRCODE_APPLICATION`

Unspecified error.

`CONF_ERRCODE_APPLICATION_INTERNAL`

As `CONF_ERRCODE_APPLICATION`, but the additional error information is only for logging/debugging, and should not be reported by northbound agents.

`CONF_ERRCODE_DATA_MISSING`

A request could not be completed because the relevant data model content does not exist.

`CONF_ERRCODE_INTERRUPT`

Processing of a request was terminated due to user interrupt - see the description of the `interrupt()` transaction callback in [confd_lib_dp\(3\)](#).

There is currently limited support for specifying one of a set of fixed error tags via `apptag_ns` and `apptag_tag`: `apptag_ns` should be 0, and `apptag_tag` can be either 0 or the hash value for a data model node.

The `fmt` and remaining arguments can specify an arbitrary string as for `confd_trans_seterr()`, but when used with one of the `code` values that has a specific meaning, it should only be given if it has

some additional information - e.g. passing "In use" with `CONFD_ERRCODE_IN_USE` is not meaningful, and will typically result in duplicated information being reported by the northbound agent. If there is no additional information, just pass an empty string ("") for *fmt*.

A call of `confd_trans_seterr(tctx, "string")` is equivalent to `confd_trans_seterr_extended(tctx, CONFD_ERRCODE_APPLICATION, 0, 0, "string")`.

When the extended error reporting is used, the northbound agents will, where possible, use the extended error information to give protocol-specific error reports to the managers, as described in the following tables. (The `CONFD_ERRCODE_INTERRUPT` code does not have a mapping here, since these interfaces do not provide the possibility to interrupt a transaction.)

For SNMP, the *code* argument is mapped to SNMP `ErrorStatus`

<i>code</i>	SNMP <i>ErrorStatus</i>
<code>CONFD_ERRCODE_IN_USE</code>	<code>resourceUnavailable</code>
<code>CONFD_ERRCODE_RESOURCE_DENIED</code>	<code>resourceUnavailable</code>
<code>CONFD_ERRCODE_INCONSISTENT_VALUE</code>	<code>inconsistentValue</code>
<code>CONFD_ERRCODE_ACCESS_DENIED</code>	<code>noAccess</code>
<code>CONFD_ERRCODE_APPLICATION</code>	<code>genErr</code>
<code>CONFD_ERRCODE_APPLICATION_INTERNAL</code>	<code>genErr</code>
<code>CONFD_ERRCODE_DATA_MISSING</code>	<code>inconsistentValue</code>

For NETCONF the *code* argument is mapped to `<error-tag>`:

<i>code</i>	NETCONF <i>error-tag</i>
<code>CONFD_ERRCODE_IN_USE</code>	<code>in-use</code>
<code>CONFD_ERRCODE_RESOURCE_DENIED</code>	<code>resource-denied</code>
<code>CONFD_ERRCODE_INCONSISTENT_VALUE</code>	<code>invalid-value</code>
<code>CONFD_ERRCODE_ACCESS_DENIED</code>	<code>access-denied</code>
<code>CONFD_ERRCODE_APPLICATION</code>	<code>operation-failed</code>
<code>CONFD_ERRCODE_APPLICATION_INTERNAL</code>	<code>operation-failed</code>
<code>CONFD_ERRCODE_DATA_MISSING</code>	<code>data-missing</code>

The tag specified by *apptag_ns/apptag_tag* will be reported as `<error-app-tag>`.

For MAAPI the *code* argument is mapped to `confd_errno`:

<i>code</i>	<i>confd_errno</i>
<code>CONFD_ERRCODE_IN_USE</code>	<code>CONFD_ERR_INUSE</code>
<code>CONFD_ERRCODE_RESOURCE_DENIED</code>	<code>CONFD_ERR_RESOURCE_DENIED</code>
<code>CONFD_ERRCODE_INCONSISTENT_VALUE</code>	<code>CONFD_ERR_INCONSISTENT_VALUE</code>
<code>CONFD_ERRCODE_ACCESS_DENIED</code>	<code>CONFD_ERR_ACCESS_DENIED</code>
<code>CONFD_ERRCODE_APPLICATION</code>	<code>CONFD_ERR_EXTERNAL</code>
<code>CONFD_ERRCODE_APPLICATION_INTERNAL</code>	<code>CONFD_ERR_APPLICATION_INTERNAL</code>
<code>CONFD_ERRCODE_DATA_MISSING</code>	<code>CONFD_ERR_DATA_MISSING</code>

The tag (if any) can be retrieved by calling

```
struct xml_tag *confd_last_error_apptag(void);
```

If no tag was provided by the callback (e.g. plain `confd_trans_seterr()` was used, or the error did not originate from a data provider callback at all), this function returns a pointer to a struct `xml_tag` with both the `ns` and the `tag` element set to 0.

In the CLI and Web UI a text string is produced through some combination of the *code* and the string given by *fmt*,

```
int confd_trans_seterr_extended_info(struct confd_trans_ctx *tctx,
enum confd_errcode code, u_int32_t apptag_ns, u_int32_t apptag_tag,
confd_tag_value_t *error_info, int n, const char *fmt, ...);
```

This function can be used to provide structured error information in the same way as `confd_trans_seterr_extended()`, and additionally provide contents for the NETCONF <error-info> element. The *error_info* argument is an array of length *n*, populated as described for the Tagged Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page. The *error_info* information is discarded for other northbound agents than NETCONF.

The `tailf:error-info` statement (see [tailf_yang_extensions\(5\)](#)) must have been used in one or more YANG modules to declare the data nodes for <error-info>. As an example, we could have this `error-info` declaration:

```
module mod {
  namespace "http://tail-f.com/test/mod";
  prefix mod;

  import tailf-common {
    prefix tailf;
  }

  ...

  tailf:error-info {
    leaf severity {
      type enumeration {
        enum info;
        enum error;
        enum critical;
      }
    }
    container detail {
      leaf class {
        type uint8;
      }
      leaf code {
        type uint8;
      }
    }
  }

  ...
}
```

A call of `confd_trans_seterr_extended_info()` to populate the <error-info> could then look like this:

```
confd_tag_value_t error_info[10];
```

```

int i = 0;

CONF_SET_TAG_ENUM_VALUE(&error_info[i],
                        mod_severity, mod_error);
CONF_SET_TAG_NS(&error_info[i], mod_ns);      i++;
CONF_SET_TAG_XMLBEGIN(&error_info[i],
                     mod_detail, mod_ns);      i++;
CONF_SET_TAG_UINT8(&error_info[i], mod_class, 42); i++;
CONF_SET_TAG_UINT8(&error_info[i], mod_code, 17); i++;
CONF_SET_TAG_XMLEND(&error_info[i],
                   mod_detail, mod_ns);      i++;
OK(confd_trans_seterr_extended_info(tctx, CONF_ERRCODE_APPLICATION,
                                   0, 0, error_info, i,
                                   "Operation failed"));

```



Note

The toplevel elements in the `confd_tag_value_t` array *must* have the `ns` element of the struct `xml_tag` set. The `CONF_SET_TAG_XMLBEGIN()` macro will set this element, but for toplevel leaf elements the `CONF_SET_TAG_NS()` macro needs to be used, as shown above.

The `<error-info>` section resulting from the above would look like this:

```

<error-info>
...
<severity xmlns="http://tail-f.com/test/mod">error</severity>
<detail xmlns="http://tail-f.com/test/mod">
  <class>42</class>
  <code>17</code>
</detail>
</error-info>

```

ERRORS

All functions in `libconfd` signal errors through the return of the `#defined CONF_ERR` - which has the value `-1` - or alternatively `CONF_EOF (-2)` which means that NSO closed its end of the socket.

Data provider callbacks (see [confd_lib_dp\(3\)](#)) can also signal errors by returning `CONF_ERR` from the callback. This can be done for all different kinds of callbacks. It is possible to provide additional error information from one of these callbacks by using one of the functions:

```

confd_trans_seterr(),      For transaction callbacks
confd_trans_seterr_extended(),
confd_trans_seterr_extended_info()
confd_db_seterr(),         For db callbacks
confd_db_seterr_extended(),
confd_db_seterr_extended_info()
confd_action_seterr(),     For action callbacks
confd_action_seterr_extended(),
confd_action_seterr_extended_info()
confd_notification_seterr(For notification callbacks
confd_notification_seterr_extended(),
confd_notification_seterr_extended_info()

```

CDB two phase subscribers (see [confd_lib_cdb\(3\)](#)) can also provide error information when `cdb_read_subscription_socket2()` has returned with type set to `CDB_SUB_PREPARE`, using one of the functions `cdb_sub_abort_trans()` and `cdb_sub_abort_trans_info()`.

Whenever `CONFID_ERR` is returned from any API function in `libconfd` it is possible to obtain additional information on the error through the symbol `confd_errno`. Additionally there may be an error text associated with the error. A call to the function

```
char *confd_lasterr(void);
```

returns a string which contains additional textual information on the error. Furthermore, the function

```
char *confd_strerror(int code);
```

returns a string which describes a particular error code. When one of the The following error codes are available:

`CONFID_ERR_NOEXISTS` (1)

Typically we tried to read a value through CDB or MAAPI which does not exist.

`CONFID_ERR_ALREADY_EXISTS` (2)

We tried to create something which already exists.

`CONFID_ERR_ACCESS_DENIED` (3)

Access to an object was denied due to AAA authorization rules.

`CONFID_ERR_NOT_WRITABLE` (4)

We tried to write an object which is not writable.

`CONFID_ERR_BADTYPE` (5)

We tried to create or write an object which is specified to have another type (see [confd_types\(3\)](#)) than the one we provided.

`CONFID_ERR_NOTCREATABLE` (6)

We tried to create an object which is not possible to create.

`CONFID_ERR_NOTDELETABLE` (7)

We tried to delete an object which is not possible to delete.

`CONFID_ERR_BADPATH` (8)

We provided a bad path in any of the `printf` style functions which take a variable number of arguments.

`CONFID_ERR_NOSTACK` (9)

We tried to pop without a preceding push.

`CONFID_ERR_LOCKED` (10)

We tried to lock something which is already locked.

`CONFID_ERR_INUSE` (11)

We tried to commit while someone else holds a lock.

`CONFID_ERR_NOTSET` (12)

A mandatory leaf does not have a value, either because it has been deleted, or not set after a create.

`CONFID_ERR_NON_UNIQUE` (13)

A group of leafs specified with the `unique` statement are not unique.

`CONFID_ERR_BAD_KEYREF` (14)

Dangling pointer.

`CONFID_ERR_TOO_FEW_ELEMS` (15)

A `min-elements` violation. A node has fewer elements or entries than specified with `min-elements`.

CONFID_ERR_TOO_MANY_ELEMS (16)
A `max-elements` violation. A node has fewer elements or entries than specified with `max-elements`.

CONFID_ERR_BADSTATE (17)
Some function, such as the MAAPI commit functions that require several functions to be called in a specific order, was called out of order.

CONFID_ERR_INTERNAL (18)
An internal error. This normally indicates a bug in NSO or libconfd (if nothing else the lack of a better error code), please report it to Tail-f support.

CONFID_ERR_EXTERNAL (19)
All errors that originate in user code.

CONFID_ERR_MALLOC (20)
Failed to allocate memory.

CONFID_ERR_PROTOUSAGE (21)
Usage of API functions or callbacks was wrong. It typically means that we invoke a function when we shouldn't. For example if we invoke the `confd_data_reply_next_key()` in a `get_elem()` callback we get this error.

CONFID_ERR_NOSESSION (22)
A session must be established prior to executing the function.

CONFID_ERR_TOOMANYTRANS (23)
A new MAAPI transaction was rejected since the transaction limit threshold was reached.

CONFID_ERR_OS (24)
An error occurred in a call to some operating system function, such as `write()`. The proper `errno` from `libc` should then be read and used as failure indicator.

CONFID_ERR_HA_CONNECT (25)
Failed to connect to a remote HA node.

CONFID_ERR_HA_CLOSED (26)
A remote HA node closed its connection to us, or there was a timeout waiting for a sync response from the master during a call of `confd_ha_beslave()`.

CONFID_ERR_HA_BADFXS (27)
A remote HA node had a different set of fxs files compared to us. It could also be that the set is the same, but the version of some fxs file is different.

CONFID_ERR_HA_BADTOKEN (28)
A remote HA node has a different token than us.

CONFID_ERR_HA_BADNAME (29)
A remote ha node has a different name than the name we think it has.

CONFID_ERR_HA_BIND (30)
Failed to bind the ha socket for incoming HA connects.

CONFID_ERR_HA_NOTICK (31)
A remote HA node failed to produce the interval live ticks.

CONFID_ERR_VALIDATION_WARNING (32)
`maapi_validate()` returned warnings.

CONFID_ERR_SUBAGENT_DOWN (33)
An operation towards a mounted NETCONF subagent failed due to the subagent not being up.

CONFID_ERR_LIB_NOT_INITIALIZED (34)
The confd library has not been properly initialized by a call to `confd_init()`.

CONFDFERRTOOMANYSESSIONS (35)

Maximum number of sessions reached.

CONFDFERRBADCONFIG (36)

An error in a configuration.

CONFDFERRRESOURCEDENIED (37)

A data provider callback returned CONFDFERRCODE_RESOURCE_DENIED (see EXTENDED ERROR REPORTING above).

CONFDFERRINCONSISTENTVALUE (38)

A data provider callback returned CONFDFERRCODE_INCONSISTENT_VALUE (see EXTENDED ERROR REPORTING above).

CONFDFERRAPPLICATIONINTERNAL (39)

A data provider callback returned CONFDFERRCODE_APPLICATION_INTERNAL (see EXTENDED ERROR REPORTING above).

CONFDFERRUNSETCHOICE (40)

No case has been selected for a mandatory choice statement.

CONFDFERRMUSTFAILED (41)

A must constraint is not satisfied.

CONFDFERRMISSINGINSTANCE (42)

The value of an instance-identifier leaf with `require-instance true` does not specify an existing instance.

CONFDFERRINVALIDINSTANCE (43)

The value of an instance-identifier leaf does not conform to the specified path filters.

CONFDFERRUNAVAILABLE (44)

We tried to use some unavailable functionality, e.g. `get/set` attributes on an operational data element.

CONFDFERREOF (45)

This value is used when a function returns CONFDFERR_EOF. Thus it is not strictly necessary to check whether the return value is CONFDFERR or CONFDFERR_EOF - if the function should return CONFDFERR_OK on success, but the return value is something else, the reason can always be found via `confd_errno`.

CONFDFERRNOTMOVABLE (46)

We tried to move an object which is not possible to move.

CONFDFERRHAWITHUPGRADE (47)

We tried to perform an in-service data model upgrade on a HA node that was either a master or a slave, or we tried to make the node a HA master or slave while an in-service data model upgrade was in progress.

CONFDFERRTIMEOUT (48)

An operation did not complete within the specified timeout.

CONFDFERRABORTED (49)

An operation was aborted.

CONFDFERRXPath (50)

Compilation or evaluation of an XPath expression failed.

CONFDFERRNOTIMPLEMENTED (51)

A request was made for an operation that wasn't implemented. This will typically occur if an application uses a version of `libconfd` that is more recent than the version of the NSO daemon, and a CDB or MAAPI function is used that is only implemented in the library version.

CONF_D_ERR_HA_BAD_VSN (52)

A remote HA node had an incompatible protocol version.

CONF_D_ERR_POLICY_FAILED (53)

A user-defined policy expression evaluated to false.

CONF_D_ERR_POLICY_COMPILATION_FAILED (54)

A user-defined policy XPath expression could not be compiled.

CONF_D_ERR_POLICY_EVALUATION_FAILED (55)

A user-defined policy expression failed XPath evaluation.

NCS_ERR_CONNECTION_REFUSED (56)

NCS failed to connect to a device.

CONF_D_ERR_START_FAILED (57)

NSO daemon failed to proceed to next start-phase.

CONF_D_ERR_DATA_MISSING (58)

A data provider callback returned CONF_D_ERRCODE_DATA_MISSING (see EXTENDED ERROR REPORTING above).

CONF_D_ERR_CLI_CMD (59)

Execution of a CLI command failed.

CONF_D_ERR_UPGRADE_IN_PROGRESS (60)

A request was made for an operation that is not allowed when in-service data model upgrade is in progress.

CONF_D_ERR_NOTRANS (61)

An invalid transaction handle was passed to a MA-API function - i.e. the handle did not refer to a transaction that was either started on, or attached to, the MA-API socket.

NCS_ERR_SERVICE_CONFLICT (62)

An NCS service invocation running outside the transaction lock modified data that was also modified by a service invocation in another transaction.

MISCELLANEOUS

The library will always set the default signal handler for SIGPIPE to be SIG_IGN. All libconfd APIs are socket based and the library must be able to detect failed write operations in a controlled manner.

The include file `confd_lib.h` includes `assert.h` and uses `assert` macros in the specialized `CONF_D_GET_XXX()` macros. If the behavior of `assert` is not wanted in a production environment, we can define `NDEBUG` before including `confd_lib.h` (or `confd.h`), see `assert(3)`. Alternatively we can define a `CONF_D_ASSERT()` macro before including `confd_lib.h`. The `assert` macros are invoked via `CONF_D_ASSERT()`, which is defined by:

```
#ifndef CONF_D_ASSERT
#define CONF_D_ASSERT(E) assert(E)
#endif
```

I.e. by defining a different version of `CONF_D_ASSERT()`, we can get our own error handler invoked instead of `assert(3)`, for example:

```
void log_error(char *file, int line, char *expr);

#define CONF_D_ASSERT(E) \
    ((E) ? (void)0 : log_error(__FILE__, __LINE__, #E))

#include <confd_lib.h>
```

SYSLOG AND DEBUG

When developing applications with `libconfd` we always need to indicate to the library which verbosity level should be used by the library. There are three different levels to choose from: `CONF_D_SILENT` where the library never writes anything, never, `CONF_D_DEBUG` where the library reports all errors and finally `CONF_D_TRACE` where the library traces the execution and invocations of all the various callback functions.

There are two different destinations for all library printouts. When we call `confd_init()`, we always need to supply a `FILE*` stream which should be used for all printouts. This parameter can be set to `NULL` if we never want any `FILE*` printouts to occur.

The second destination is syslog, i.e. the library will syslog if told to. This is controlled by the global integer variable `confd_lib_use_syslog`. If we set this variable to 1, `libconfd` will syslog all output. If we set it to 0 the library will not syslog. It is the responsibility of the application to (optionally) call `openlog()` before initializing the NSO library. The default value is 0.

There also exists a hook point at which a library user can install their own printer. This is done by assigning to a global variable `confd_user_log_hook`, as in:

```
void mylogger(int syslogprio, const char *fmt, va_list ap) {
    char buf[BUFSIZ];
    sprintf(buf, "MYLOG:(%d) ", syslogprio); strcat(buf, fmt);
    vfprintf(stderr, buf, ap);
}
```

```
confd_user_log_hook = mylogger;
```

The *syslogprio* is `LOG_ERR` or `LOG_CRIT` for error messages, and `LOG_DEBUG` for trace messages, see the description of `confd_init()`.

Thus a good combination of values in a target environment is to set the `FILE*` handle to `NULL` and `confd_lib_use_syslog` to 1. This way we do not get the overhead of file logging and at the same time get all errors reported to syslog.

SEE ALSO

`ncs(5)` - NSO daemon configuration file format

The NSO User Guide

Name

confd_lib_maapi — MAAPI (Management Agent API). A library for connecting to NCS with a read/write interface inside transactions.

Synopsis

```
#include <confd_lib.h> #include <confd_maapi.h>

int maapi_start_user_session(int sock, const char *username, const char
*context, const char **groups, int numgroups, const struct confd_ip
*src_addr, enum confd_proto prot);

int maapi_start_user_session2(int sock, const char *username, const
char *context, const char **groups, int numgroups, const struct
confd_ip *src_addr, int src_port, enum confd_proto prot);

int maapi_start_trans(int sock, enum confd_dbname dbname, enum
confd_trans_mode readwrite);

int maapi_start_trans2(int sock, enum confd_dbname dbname, enum
confd_trans_mode readwrite, int usid);

int maapi_start_trans_flags(int sock, enum confd_dbname dbname, enum
confd_trans_mode readwrite, int usid, int flags);

int maapi_connect(int sock, const struct sockaddr* srv, int srv_sz);

int maapi_load_schemas(int sock);

int maapi_load_schemas_list(int sock, int flags, const u_int32_t
*nshash, const int *nsflags, int num_ns);

int maapi_get_schema_file_path(int sock, char **buf);

int maapi_close(int sock);

int maapi_start_user_session3(int sock, const char *username, const
char *context, const char **groups, int numgroups, const struct
confd_ip *src_addr, int src_port, enum confd_proto prot, const
char *vendor, const char *product, const char *version, const char
*client_id);

int maapi_end_user_session(int sock);

int maapi_kill_user_session(int sock, int usessid);

int maapi_get_user_sessions(int sock, int res[], int n);

int maapi_get_user_session(int sock, int usessid, struct
confd_user_info *us);

int maapi_get_my_user_session_id(int sock);

int maapi_set_user_session(int sock, int usessid);
```

```

int maapi_get_user_session_identification(int sock, int usessid, struct
confd_user_identification *uident);

int maapi_get_user_session_opaque(int sock, int usessid, char
**opaque);

int maapi_get_authorization_info(int sock, int usessid, struct
confd_authorization_info **ainfo);

int maapi_set_next_user_session_id(int sock, int usessid);

int maapi_lock(int sock, enum confd_dbname name);

int maapi_unlock(int sock, enum confd_dbname name);

int maapi_is_lock_set(int sock, enum confd_dbname name);

int maapi_lock_partial(int sock, enum confd_dbname name, char
*xpaths[], int nxpaths, int *lockid);

int maapi_unlock_partial(int sock, int lockid);

int maapi_candidate_validate(int sock);

int maapi_delete_config(int sock, enum confd_dbname name);

int maapi_candidate_commit(int sock);

int maapi_candidate_commit_persistent(int sock, const char
*persist_id);

int maapi_candidate_commit_info(int sock, const char *persist_id, const
char *label, const char *comment);

int maapi_candidate_commit_persistent_flags(int sock, const char
*persist_id, int flags);

int maapi_candidate_confirmed_commit(int sock, int timeoutsecs);

int maapi_candidate_confirmed_commit_persistent(int sock, int
timeoutsecs, const char *persist, const char *persist_id);

int maapi_candidate_confirmed_commit_info(int sock, int timeoutsecs,
const char *persist, const char *persist_id, const char *label, const
char *comment);

int maapi_candidate_confirmed_commit_persistent_flags(int sock, int
timeoutsecs, const char *persist, const char *persist_id, int flags);

int maapi_candidate_abort_commit(int sock);

int maapi_candidate_abort_commit_persistent(int sock, const char
*persist_id);

int maapi_candidate_reset(int sock);

```

```

int maapi_confirmed_commit_in_progress(int sock);

int maapi_copy_running_to_startup(int sock);

int maapi_is_running_modified(int sock);

int maapi_is_candidate_modified(int sock);

int maapi_start_trans_flags2(int sock, enum confd_dbname dbname, enum
confd_trans_mode readwrite, int usid, int flags, const char *vendor,
const char *product, const char *version, const char *client_id);

int maapi_start_trans_in_trans(int sock, enum confd_trans_mode
readwrite, int usid, int thandle);

int maapi_finish_trans(int sock, int thandle);

int maapi_validate_trans(int sock, int thandle, int unlock, int
forcevalidation);

int maapi_prepare_trans(int sock, int thandle);

int maapi_prepare_trans_flags(int sock, int thandle, int flags);

int maapi_commit_trans(int sock, int thandle);

int maapi_abort_trans(int sock, int thandle);

int maapi_apply_trans(int sock, int thandle, int keepopen);

int maapi_apply_trans_flags(int sock, int thandle, int keepopen, int
flags);

int maapi_commit_queue_result(int sock, int thandle, int timeoutsecs,
struct ncs_commit_queue_result *result);

int maapi_set_namespace(int sock, int thandle, int hashed_ns);

int maapi_cd(int sock, int thandle, const char *fmt, ...);

int maapi_pushd(int sock, int thandle, const char *fmt, ...);

int maapi_popd(int sock, int thandle);

int maapi_getcwd(int sock, int thandle, size_t strsz, char *curdir);

int maapi_getcwd_kpath(int sock, int thandle, confd_hkeypath_t **kp);

int maapi_exists(int sock, int thandle, const char *fmt, ...);

int maapi_num_instances(int sock, int thandle, const char *fmt, ...);

int maapi_get_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, ...);

int maapi_get_int8_elem(int sock, int thandle, int8_t *rval, const char
*fmt, ...);

```

```

int maapi_get_int16_elem(int sock, int thandle, int16_t *rval, const
char *fmt, ...);

int maapi_get_int32_elem(int sock, int thandle, int32_t *rval, const
char *fmt, ...);

int maapi_get_int64_elem(int sock, int thandle, int64_t *rval, const
char *fmt, ...);

int maapi_get_u_int8_elem(int sock, int thandle, u_int8_t *rval, const
char *fmt, ...);

int maapi_get_u_int16_elem(int sock, int thandle, u_int16_t *rval,
const char *fmt, ...);

int maapi_get_u_int32_elem(int sock, int thandle, u_int32_t *rval,
const char *fmt, ...);

int maapi_get_u_int64_elem(int sock, int thandle, u_int64_t *rval,
const char *fmt, ...);

int maapi_get_ipv4_elem(int sock, int thandle, struct in_addr *rval,
const char *fmt, ...);

int maapi_get_ipv6_elem(int sock, int thandle, struct in6_addr *rval,
const char *fmt, ...);

int maapi_get_double_elem(int sock, int thandle, double *rval, const
char *fmt, ...);

int maapi_get_bool_elem(int sock, int thandle, int *rval, const char
*fmt, ...);

int maapi_get_datetime_elem(int sock, int thandle, struct
confd_datetime *rval, const char *fmt, ...);

int maapi_get_date_elem(int sock, int thandle, struct confd_date *rval,
const char *fmt, ...);

int maapi_get_time_elem(int sock, int thandle, struct confd_time *rval,
const char *fmt, ...);

int maapi_get_duration_elem(int sock, int thandle, struct
confd_duration *rval, const char *fmt, ...);

int maapi_get_enum_value_elem(int sock, int thandle, int32_t *rval,
const char *fmt, ...);

int maapi_get_bit32_elem(int sock, int thandle, u_int32_t *rval, const
char *fmt, ...);

int maapi_get_bit64_elem(int sock, int thandle, u_int64_t *rval, const
char *fmt, ...);

int maapi_get_bitbig_elem(int sock, int thandle, unsigned char **rval,
int *bufsiz, const char *fmt, ...);

```

```

int maapi_get_objectref_elem(int sock, int thandle, confd_hkeypath_t
**rval, const char *fmt, ...);

int maapi_get_oid_elem(int sock, int thandle, struct confd_snmp_oid
**rval, const char *fmt, ...);

int maapi_get_buf_elem(int sock, int thandle, unsigned char **rval, int
*bufsiz, const char *fmt, ...);

int maapi_get_str_elem(int sock, int thandle, char *buf, int n, const
char *fmt, ...);

int maapi_get_binary_elem(int sock, int thandle, unsigned char **rval,
int *bufsiz, const char *fmt, ...);

int maapi_get_hexstr_elem(int sock, int thandle, unsigned char **rval,
int *bufsiz, const char *fmt, ...);

int maapi_get_qname_elem(int sock, int thandle, unsigned char
**prefix, int *prefixsz, unsigned char **name, int *namesz, const char
*fmt, ...);

int maapi_get_list_elem(int sock, int thandle, confd_value_t **values,
int *n, const char *fmt, ...);

int maapi_get_ipv4prefix_elem(int sock, int thandle, struct
confd_ipv4_prefix *rval, const char *fmt, ...);

int maapi_get_ipv6prefix_elem(int sock, int thandle, struct
confd_ipv6_prefix *rval, const char *fmt, ...);

int maapi_get_decimal64_elem(int sock, int thandle, struct
confd_decimal64 *rval, const char *fmt, ...);

int maapi_get_identityref_elem(int sock, int thandle, struct
confd_identityref *rval, const char *fmt, ...);

int maapi_get_ipv4_and_plen_elem(int sock, int thandle, struct
confd_ipv4_prefix *rval, const char *fmt, ...);

int maapi_get_ipv6_and_plen_elem(int sock, int thandle, struct
confd_ipv6_prefix *rval, const char *fmt, ...);

int maapi_get_dquad_elem(int sock, int thandle, struct
confd_dotted_quad *rval, const char *fmt, ...);

int maapi_vget_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, va_list args);

int maapi_init_cursor(int sock, int thandle, struct maapi_cursor *mc,
const char *fmt, ...);

int maapi_get_next(struct maapi_cursor *mc);

int maapi_find_next(struct maapi_cursor *mc, enum confd_find_next_type
type, confd_value_t *inkeys, int n_inkeys);

```

```

void maapi_destroy_cursor(struct maapi_cursor *mc);

int maapi_set_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, ...);

int maapi_set_elem2(int sock, int thandle, const char *strval, const
char *fmt, ...);

int maapi_vset_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, va_list args);

int maapi_create(int sock, int thandle, const char *fmt, ...);

int maapi_delete(int sock, int thandle, const char *fmt, ...);

int maapi_get_object(int sock, int thandle, confd_value_t *values, int
n, const char *fmt, ...);

int maapi_get_objects(struct maapi_cursor *mc, confd_value_t *values,
int n, int *nobj);

int maapi_get_values(int sock, int thandle, confd_tag_value_t *values,
int n, const char *fmt, ...);

int maapi_set_object(int sock, int thandle, const confd_value_t
*values, int n, const char *fmt, ...);

int maapi_set_values(int sock, int thandle, const confd_tag_value_t
*values, int n, const char *fmt, ...);

int maapi_get_case(int sock, int thandle, const char *choice,
confd_value_t *rcase, const char *fmt, ...);

int maapi_get_attrs(int sock, int thandle, u_int32_t *attrs, int
num_attrs, confd_attr_value_t **attr_vals, int *num_vals, const char
*fmt, ...);

int maapi_set_attr(int sock, int thandle, u_int32_t attr, confd_value_t
*v, const char *fmt, ...);

int maapi_delete_all(int sock, int thandle, enum maapi_delete_how how);

int maapi_revert(int sock, int thandle);

int maapi_set_flags(int sock, int thandle, int flags);

int maapi_set_delayed_when(int sock, int thandle, int on);

int maapi_set_label(int sock, int thandle, const char *label);

int maapi_set_comment(int sock, int thandle, const char *comment);

int maapi_copy(int sock, int from_thandle, int to_thandle);

int maapi_copy_path(int sock, int from_thandle, int to_thandle, const
char *fmt, ...);

```

```

int maapi_copy_tree(int sock, int thandle, const char *from, const char
*tofmt, ...);

int maapi_insert(int sock, int thandle, const char *fmt, ...);

int maapi_move(int sock, int thandle, confd_value_t* tokey, int n,
const char *fmt, ...);

int maapi_move_ordered(int sock, int thandle, enum maapi_move_where
where, confd_value_t* tokey, int n, const char *fmt, ...);

int maapi_shared_create(int sock, int thandle, int flags, const char
*fmt, ...);

int maapi_shared_set_elem(int sock, int thandle, confd_value_t *v, int
flags, const char *fmt, ...);

int maapi_shared_set_elem2(int sock, int thandle, const char *strval,
int flags, const char *fmt, ...);

int maapi_shared_set_values(int sock, int thandle, const
confd_tag_value_t *values, int n, int flags, const char *fmt, ...);

int maapi_shared_insert(int sock, int thandle, int flags, const char
*fmt, ...);

int maapi_shared_copy_tree(int sock, int thandle, int flags, const char
*from, const char *tofmt, ...);

int maapi_ncs_apply_template(int sock, int thandle, char
*template_name, const struct ncs_name_value *variables, int
num_variables, int flags, const char *rootfmt, ...);

int maapi_shared_ncs_apply_template(int sock, int thandle, char
*template_name, const struct ncs_name_value *variables, int
num_variables, int flags, const char *rootfmt, ...);

int maapi_ncs_get_templates(int sock, char ***templates, int
*num_templates);

int maapi_ncs_write_service_log_entry(int sock, const char *msg,
confd_value_t *type, confd_value_t *level, const char *fmt, ...);

int maapi_authenticate(int sock, const char *user, const char *pass,
char *groups[], int n);

int maapi_authenticate2(int sock, const char *user, const char *pass,
const struct confd_ip *src_addr, int src_port, const char *context,
enum confd_proto prot, char *groups[], int n);

int maapi_attach(int sock, int hashed_ns, struct confd_trans_ctx *ctx);

int maapi_attach2(int sock, int hashed_ns, int usid, int thandle);

int maapi_attach_init(int sock, int *thandle);

```

```

int maapi_detach(int sock, struct confd_trans_ctx *ctx);

int maapi_detach2(int sock, int thandle);

int maapi_diff_iterate(int sock, int thandle, enum maapi_iter_ret
(*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op op, confd_value_t
*oldv, confd_value_t *newv, void *state), int flags, void *initstate);

int maapi_keypath_diff_iterate(int sock, int thandle, enum
maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op op,
confd_value_t *oldv, confd_value_t *newv, void *state), int flags, void
*initstate, const char *fmtpath, ...);

int maapi_diff_iterate_resume(int sock, enum maapi_iter_ret reply,
enum maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op
op, confd_value_t *oldv, confd_value_t *newv, void *state), void
*resumestate);

int maapi_iterate(int sock, int thandle, enum maapi_iter_ret (*iter, )
(confd_hkeypath_t *kp, confd_value_t *v, confd_attr_value_t *attr_vals,
int num_attr_vals, void *state), int flags, void *initstate, const char
*fmtpath, ...);

int maapi_iterate_resume(int sock, enum maapi_iter_ret reply, enum
maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, confd_value_t *v,
confd_attr_value_t *attr_vals, int num_attr_vals, void *state), void
*resumestate);

int maapi_get_running_db_status(int sock);

int maapi_set_running_db_status(int sock, int status);

int maapi_list_rollbacks(int sock, struct maapi_rollback *rp, int
*rp_size);

int maapi_load_rollback(int sock, int thandle, int rollback_num);

int maapi_request_action(int sock, confd_tag_value_t *params, int
nparams, confd_tag_value_t **values, int *nvalues, int hashed_ns, const
char *fmt, ...);

int maapi_request_action_th(int sock, int thandle, confd_tag_value_t
*params, int nparams, confd_tag_value_t **values, int *nvalues, const
char *fmt, ...);

int maapi_request_action_str_th(int sock, int thandle, char **output,
const char *cmd_fmt, const char *path_fmt, ...);

int maapi_xpath2kpath(int sock, const char *xpath, confd_hkeypath_t
**hkp);

int maapi_user_message(int sock, const char *to, const char *message,
const char *sender);

int maapi_sys_message(int sock, const char *to, const char *message);

```



```

int maapi_prio_message(int sock, const char *to, const char *message);

int maapi_cli_diff_cmd(int sock, int thandle, int thandle_old, char
*res, int size, int flags, const char *fmt, ...);

int maapi_cli_accounting(int sock, const char *user, const int usid,
const char *cmdstr);

int maapi_cli_path_cmd(int sock, int thandle, char *res, int size, int
flags, const char *fmt, ...);

int maapi_cli_cmd_to_path(int sock, const char *line, char *ns, int
nsize, char *path, int psize);

int maapi_cli_cmd_to_path2(int sock, int thandle, const char *line,
char *ns, int nsize, char *path, int psize);

int maapi_cli_prompt(int sock, int usess, const char *prompt, int echo,
char *res, int size);

int maapi_cli_prompt2(int sock, int usess, const char *prompt, int
echo, int timeout, char *res, int size);

int maapi_cli_prompt_oneof(int sock, int usess, const char *prompt,
char **choice, int count, char *res, int size);

int maapi_cli_prompt_oneof2(int sock, int usess, const char *prompt,
char **choice, int count, int timeout, char *res, int size);

int maapi_cli_read_eof(int sock, int usess, int echo, char *res, int
size);

int maapi_cli_read_eof2(int sock, int usess, int echo, int timeout,
char *res, int size);

int maapi_cli_write(int sock, int usess, const char *buf, int size);

int maapi_cli_cmd(int sock, int usess, const char *buf, int size);

int maapi_cli_cmd2(int sock, int usess, const char *buf, int size, int
flags);

int maapi_cli_cmd3(int sock, int usess, const char *buf, int size, int
flags, const char *unhide, int usize);

int maapi_cli_cmd4(int sock, int usess, const char *buf, int size, int
flags, char **unhide, int usize);

int maapi_cli_cmd_io(int sock, int usess, const char *buf, int size,
int flags, const char *unhide, int usize);

int maapi_cli_cmd_io2(int sock, int usess, const char *buf, int size,
int flags, char **unhide, int usize);

int maapi_cli_cmd_io_result(int sock, int id);

```

```

int maapi_cli_printf(int sock, int usess, const char *fmt, ...);

int maapi_cli_vprintf(int sock, int usess, const char *fmt, va_list
args);

int maapi_cli_set(int sock, int usess, const char *opt, const char
*value);

int maapi_cli_get(int sock, int usess, const char *opt, char *res, int
size);

int maapi_set_readonly_mode(int sock, int flag);

int maapi_disconnect_remote(int sock, const char *address);

int maapi_disconnect_sockets(int sock, int *sockets, int nsocks);

int maapi_save_config(int sock, int thandle, int flags, const char
*fmtpath, ...);

int maapi_save_config_result(int sock, int id);

int maapi_load_config(int sock, int thandle, int flags, const char
*filename);

int maapi_load_config_cmds(int sock, int thandle, int flags, const char
*cmds, const char *fmt, ...);

int maapi_load_config_stream(int sock, int thandle, int flags);

int maapi_load_config_stream_result(int sock, int id);

int maapi_roll_config(int sock, int thandle, const char *fmtpath, ...);

int maapi_roll_config_result(int sock, int id);

int maapi_get_stream_progress(int sock, int id);

int maapi_xpath_eval(int sock, int thandle, const char *expr, int
(*result, )(confd_hkeypath_t *kp, confd_value_t *v, void *state), void
(*trace)(char *), void *initstate, const char *fmtpath, ...);

int maapi_xpath_eval_expr(int sock, int thandle, const char *expr, char
**res, void (*trace)(char *), const char *fmtpath, ...);

int maapi_query_start(int sock, int thandle, const char *expr,
const char *context_node, int chunk_size, int initial_offset, enum
confd_query_result_type result_as, int nselect, const char *select[],
int nsort, const char *sort[]);

int maapi_query_startv(int sock, int thandle, const char *expr,
const char *context_node, int chunk_size, int initial_offset, enum
confd_query_result_type result_as, int select_nparams, ...);

int maapi_query_result(int sock, int gh, struct confd_query_result
**qrs);

```

```

int maapi_query_result_count(int sock, int qh);

int maapi_query_free_result(struct confd_query_result *qrs);

int maapi_query_reset_to(int sock, int qh, int offset);

int maapi_query_reset(int sock, int qh);

int maapi_query_stop(int sock, int qh);

int maapi_do_display(int sock, int thandle, const char *fmtpath, ...);

int maapi_install_crypto_keys(int sock);

int maapi_init_upgrade(int sock, int timeoutsecs, int flags);

int maapi_perform_upgrade(int sock, const char **loadpathdirs, int n);

int maapi_commit_upgrade(int sock);

int maapi_abort_upgrade(int sock);

int maapi_aaa_reload(int sock, int synchronous);

int maapi_aaa_reload_path(int sock, int synchronous, const char
*fmt, ...);

int maapi_start_phase(int sock, int phase, int synchronous);

int maapi_wait_start(int sock, int phase);

int maapi_reload_config(int sock);

int maapi_reopen_logs(int sock);

int maapi_stop(int sock, int synchronous);

int maapi_rebind_listener(int sock, int listener);

int maapi_clear_opcache(int sock, const char *fmt, ...);

```

LIBRARY

NCS Library, (libconfd, -lconfd)

DESCRIPTION

The libconfd shared library is used to connect to the NSO transaction manager. The API described in this man page has several purposes. We can use MAAPI when we wish to implement our own proprietary management agent. We also use MAAPI to attach to already existing NSO transactions, for example when we wish to implement semantic validation of configuration data in C, and also when we wish to implement CLI wizards in C.

PATHS

The majority of the functions described here take as their two last arguments a format string and a variable number of extra arguments as in: `char *fmt, ...`;

The paths for MAAPI work like paths for CDB (see [confd_lib_cdb\(3\)](#)) with the exception that the bracket notation '[n]' is not allowed for MAAPI paths.

All the functions that take a path on this form also have a `va_list` variant, of the same form as `maapi_vget_elem()` and `maapi_vset_elem()`, which are the only ones explicitly documented below. I.e. they have a prefix "maapi_v" instead of "maapi_", and take a single `va_list` argument instead of a variable number of arguments.

FUNCTIONS

All functions return `CONFDF_OK` (0), `CONFDF_ERR` (-1) or `CONFDF_EOF` (-2) unless otherwise stated. Whenever `CONFDF_ERR` is returned from any API function in `confd_lib_maapi` it is possible to obtain additional information on the error through the symbol `confd_errno`, see the `ERRORS` section of [confd_lib_lib\(3\)](#).

In the case of `CONFDF_EOF` it means that the socket to NCS has been closed.

```
int maapi_connect(int sock, const struct sockaddr* srv, int srv_sz);
```

The application has to connect to NCS before it can interact with NCS.



Note

If this call fails (i.e. does not return `CONFDF_OK`), the socket descriptor must be closed and a new socket created before the call is re-attempted.

Errors: `CONFDF_ERR_MALLOC`, `CONFDF_ERR_OS`

```
int maapi_load_schemas(int sock);
```

This function dynamically loads schema information from the NSO daemon into the library, where it is available to all the library components as described in the [confd_types\(3\)](#) and [confd_lib_lib\(3\)](#) man pages. See also `confd_load_schemas()` in [confd_lib_lib\(3\)](#).

Errors: `CONFDF_ERR_MALLOC`, `CONFDF_ERR_OS`

```
int maapi_load_schemas_list(int sock, int flags, const u_int32_t  
*nshash, const int *nsflags, int num_ns);
```

A variant of `maapi_load_schemas()` that allows for loading a subset of the schema information from the NSO daemon into the library. This means that the loading can be significantly faster in the case of a system with many large data models, with the drawback that the functions that use the schema information will have limited functionality or not work at all.

The `flags` parameter can be given as `CONFDF_LOAD_SCHEMA_HASH` to request that the global mapping between strings and hash values for the data model nodes should be loaded. If `flags` is given as 0, this mapping is not loaded. The mapping is required for use of the functions `confd_hash2str()`, `confd_str2hash()`, `confd_cs_node_cd()`, and `confd_xpath_pp_kpath()`. Additionally, without the mapping, `confd_pp_value()`, `confd_pp_kpath()`, and `confd_pp_kpath_len()`, as well as the trace printouts from the library, will print nodes as "tag<N>", where N is the hash value, instead of the node name.

The `nshash` parameter is a `num_ns` elements long array of namespace hash values, requesting that schema information should be loaded for the listed namespaces according to the corresponding element of

the *nsflags* array (also *num_ns* elements long). For each namespace, either or both of these flags may be given:

- CONFID_LOAD_SCHEMA_NODES** This flag requests that the *confd_cs_node* tree (see [confd_types\(3\)](#)) for the namespace should be loaded. This tree is required for the use of the functions *confd_find_cs_root()*, *confd_find_cs_node()*, *confd_find_cs_node_child()*, *confd_cs_node_cd()*, *confd_register_node_type()*, *confd_get_leaf_list_type()*, and *confd_xpath_pp_kpath()* for the namespace. Additionally, the above functions that print a *confd_hkeypath_t*, as well as the library trace printouts, will attempt to use this tree and the type information (see below) to find the correct string representation for key values - if the tree isn't available, key values will be printed as described for *confd_pp_value()*.
- CONFID_LOAD_SCHEMA_TYPES** This flag requests that information about the types defined in the namespace should be loaded. The type information is required for use of the functions *confd_val2str()*, *confd_str2val()*, *confd_find_ns_type()*, *confd_get_leaf_list_type()*, *confd_register_ns_type()*, and *confd_register_node_type()* for the namespace. Additionally the *confd_hkeypath_t*-printing functions and the library trace printouts will also fall back to *confd_pp_value()* as described above if the type information isn't available.
- Type definitions may refer to types defined in other namespaces. If the **CONFID_LOAD_SCHEMA_TYPES** flag has been given for a namespace, and the types defined there have such type references to namespaces that are not included in the *nshash* array, the referenced type information will also be loaded, if necessary recursively, until the types have a complete definition.

See also *confd_load_schemas_list()* in [confd_lib_lib\(3\)](#).

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS

```
int maapi_get_schema_file_path(int sock, char **buf);
```

If shared memory schema support has been enabled via */ncs-config/enable-shared-memory-schema* in *ncs.conf*, this function will return the pathname of the file used for the shared memory mapping, which can then be passed to *confd_mmap_schemas()* (see [confd_lib_lib\(3\)](#)). If the call is successful, *buf* is set to point to a dynamically allocated string, which must be freed by the application by means of calling *free(3)*.

If creation of the schema file is in progress when the function is called, the call will block until the creation has completed. If shared memory schema support has not been enabled, or if the creation of the schema file failed, the function returns CONFID_ERR with *confd_errno* set to CONFID_ERR_NOEXISTS.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOEXISTS

```
int maapi_close(int sock);
```

Effectively a call to *maapi_end_user_session()* and also closes the socket.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION

Even if the call returns an error, the socket will be closed.

SESSION MANAGEMENT

```
int maapi_start_user_session(int sock, const char *username, const char
*context, const char **groups, int numgroups, const struct confd_ip
*src_addr, enum confd_proto prot);
```

Once we have created a MAAPI socket, we must also establish a user session on the socket. It is up to the user of the MAAPI library to authenticate users. The library user can ask NCS to perform the actual authentication through a call to `maapi_authenticate()` but authentication may very well occur through some other external means.

Thus, when we use this function to create a user session, we must provide all relevant information about the user. If we wish to execute read/write transactions over the MAAPI interface, we must first have an established user session.

A user session corresponds to a NETCONF manager who has just established an authenticated SSH connection, but not yet sent any NETCONF commands on the SSH connection.

The struct `confd_ip` is defined in `confd_lib.h` and must be properly populated before the call. For example:

```
struct confd_ip ip;
ip.af = AF_INET;
inet_aton("10.0.0.33", &ip.ip.v4);
```

The `context` parameter can be any string. The string provided here is precisely the context string which will be used to authorize all data access through the AAA system. Each AAA rule has a context string which must match in order for a AAA rule to match. (See the AAA chapter in the User Guide.)

Using the string "system" for `context` has special significance:

- The session is exempt from all `maxSessions` limits in `confd.conf`.
- There will be no authorization checks done by the AAA system.
- The session is not logged in the audit log.
- The session is not shown in 'confd --status', nor 'show users' in CLI etc.
- The session may be started already in NCS start phase 0. (However read-write transactions can not be started until phase 1, i.e. transactions started in phase 0 must use parameter `readwrite == CONFD_READ`).

Thus this can be useful e.g. when we need to create the user session for an "internal" transaction done by an application, without relation to a session from a northbound agent. Of course the implications of the above need to be carefully considered in each case.

It is not possible to create new user sessions until NSO has reached start phase 2 (See [confd\(1\)](#)), with the above exception of a session with the context set to "system".

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_ALREADY_EXISTS, CONFD_ERR_BADSTATE

```
int maapi_start_user_session2(int sock, const char *username, const
char *context, const char **groups, int numgroups, const struct
confd_ip *src_addr, int src_port, enum confd_proto prot);
```

This function does the same as `maapi_start_user_session()`, but allows for the TCP/UDP source port to be passed to NCS. Calling `maapi_start_user_session()` is equivalent to calling `maapi_start_user_session2()` with `src_port` 0.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_ALREADY_EXISTS, CONFD_ERR_BADSTATE

```
int maapi_start_user_session3(int sock, const char *username, const
char *context, const char **groups, int numgroups, const struct
confd_ip *src_addr, int src_port, enum confd_proto prot, const
char *vendor, const char *product, const char *version, const char
*client_id);
```

This function does the same as `maapi_start_user_session2()`, but allows additional information about the session to be passed to NCS. Calling `maapi_start_user_session2()` is equivalent to calling `maapi_start_user_session3()` with `vendor`, `product` and `version` set to NULL, and `client_id` set to `__MAAPI_CLIENT_ID__`. The `__MAAPI_CLIENT_ID__` macro (defined in `confd_maapi.h`) will expand to a string representation of `__FILE__:__LINE__`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_ALREADY_EXISTS, CONFD_ERR_BADSTATE

```
int maapi_end_user_session(int sock);
```

Ends our own user session. If the MAAPI socket is closed, the user session is automatically ended.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION

```
int maapi_kill_user_session(int sock, int usessid);
```

Kill the user session identified by `usessid`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_get_user_sessions(int sock, int res[], int n);
```

Get the `usessid` for all current user sessions. The `res` array is populated with at most `n` `usessids`, and the total number of user sessions is returned (i.e. if the return value is larger than `n`, the array was too short to hold all `usessids`).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

```
int maapi_get_user_session(int sock, int usessid, struct
confd_user_info *us);
```

Populate the `confd_user_info` structure with the data for the user session identified by `usessid`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_get_my_user_session_id(int sock);
```

A user session is identified through an integer index, a `usessid`. This function returns the `usessid` associated with the MAAPI socket `sock`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_set_user_session(int sock, int usessid);
```

Associate the socket with an already existing user session. This can be used instead of `maapi_start_user_session()` when we really do not want to start a new user session, e.g. if we want to call an action on behalf of a given user session.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_get_user_session_identification(int sock, int usessid, struct  
confd_user_identification *uident);
```

If the flag `CONFD_USESS_FLAG_HAS_IDENTIFICATION` is set in the `flags` field of the `confd_user_info` structure, additional identification information has been provided by the northbound client. This information can then be retrieved into a `confd_user_identification` structure (see `confd_lib.h`) by calling this function. The elements of `confd_user_identification` are either `NULL` (if the corresponding information was not provided) or point to a string. The strings must be freed by the application by means of calling `free(3)`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_get_user_session_opaque(int sock, int usessid, char  
**opaque);
```

If the flag `CONFD_USESS_FLAG_HAS_OPAQUE` is set in the `flags` field of the `confd_user_info` structure, "opaque" information has been provided by the northbound client (see the `-O` option in [confd_cli\(1\)](#)). The information can then be retrieved by calling this function. If the call is successful, `opaque` is set to point to a dynamically allocated string, which must be freed by the application by means of calling `free(3)`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_get_authorization_info(int sock, int usessid, struct  
confd_authorization_info **ainfo);
```

This function retrieves authorization info for a user session, i.e. the groups that the user has been assigned to. The struct `confd_authorization_info` is defined as:

```
struct confd_authorization_info {  
    int ngroups;  
    char **groups;  
};
```

If the call is successful, `ainfo` is set to point to a dynamically allocated structure, which must be freed by the application by means of calling `confd_free_authorization_info()` (see [confd_lib_lib\(3\)](#)).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_set_next_user_session_id(int sock, int usessid);
```

Set the user session id that will be assigned to the next user session started. The given value is silently forced to be in the range $100 \dots 2^{31}-1$. This function can be used to ensure that session ids for user sessions started by northbound agents or via MAAPI are unique across a NCS restart.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS

LOCKS

```
int maapi_lock(int sock, enum confd_dbname name);
```

```
int maapi_unlock(int sock, enum confd_dbname name);
```

These functions can be used to manipulate locks on the 3 different database types. If `maapi_lock()` is called and the database is already locked, `CONFD_ERR` is returned, and `confd_errno` will be set to `CONFD_ERR_LOCKED`. If `confd_errno` is `CONFD_ERR_EXTERNAL` it means that a callback has been invoked in an external database to lock/unlock which in its turn returned an error. (See [confd_lib_dp\(3\)](#) for external database callback API)

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_LOCKED`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_NOSESSION`

```
int maapi_is_lock_set(int sock, enum confd_dbname name);
```

Returns a positive integer being the `usid` of the current lock owner if the lock is set, and 0 if the lock is not set.

```
int maapi_lock_partial(int sock, enum confd_dbname name, char  
*xpaths[], int nxpaths, int *lockid);
```

```
int maapi_unlock_partial(int sock, int lockid);
```

We can also manipulate partial locks on the databases, i.e. locks on a specified set of leafs and/or subtrees. The specification of what to lock is given via the `xpaths` array, which is populated with `nxpaths` pointers to XPath expressions. If the lock succeeds, `maapi_lock_partial()` returns `CONFD_OK`, and a lock identifier to use with `maapi_unlock_partial()` is stored in `*lockid`.

If `CONFD_ERR` is returned, some values of `confd_errno` are of particular interest:

`CONFD_ERR_LOCKED` Some of the requested nodes are already locked.

`CONFD_ERR_EXTERNAL` A callback has been invoked in an external database to `lock_partial/unlock_partial` which in its turn returned an error (see [confd_lib_dp\(3\)](#) for external database callback API).

`CONFD_ERR_NOEXISTS` The list of XPath expressions evaluated to an empty set of nodes - i.e. there is nothing to lock.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_LOCKED`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`

CANDIDATE MANIPULATION

All the candidate manipulation functions require that the candidate data store is enabled in `confd.conf` - otherwise they will set `confd_errno` to `CONFD_ERR_NOEXISTS`. If the candidate data store is enabled, `confd_errno` may be set to `CONFD_ERR_NOEXISTS` for other reasons, as described below.

All these functions may also set `confd_errno` to `CONFD_ERR_EXTERNAL`. This value can only be set when the candidate is owned by the external database. When NCS owns the candidate, which is the most common configuration scenario, the candidate manipulation function will never set `confd_errno` to `CONFD_ERR_EXTERNAL`.

```
int maapi_candidate_validate(int sock);
```

This function validates the candidate. The function should only be used when the candidate is not owned by NCS, i.e. when the candidate is owned by an external database.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_EXTERNAL

```
int maapi_candidate_commit(int sock);
```

This function copies the candidate to running. It is also used to confirm a previous call to `maapi_candidate_confirmed_commit()`, i.e. to prevent the automatic rollback if a confirmed commit is not confirmed.

If `confd_errno` is `CONFD_ERR_INUSE` it means that some other user session is doing a confirmed commit or has a lock on the database. `CONFD_ERR_NOEXISTS` means that there is an ongoing persistent confirmed commit (see below) - i.e. there is no confirmed commit that this function call can apply to.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS, CONFD_ERR_INUSE, CONFD_ERR_NOSESSION, CONFD_ERR_EXTERNAL

```
int maapi_candidate_confirmed_commit(int sock, int timeoutsecs);
```

This function also copies the candidate into running. However if a call to `maapi_candidate_commit()` is not done within `timeoutsecs` an automatic rollback will occur. It can also be used to "extend" a confirmed commit that is already in progress, i.e. set a new timeout or add changes.

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit (see below).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS, CONFD_ERR_INUSE, CONFD_ERR_NOSESSION, CONFD_ERR_EXTERNAL

```
int maapi_candidate_abort_commit(int sock);
```

This function cancels an ongoing confirmed commit.

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that some other user session initiated the confirmed commit, or that there is an ongoing persistent confirmed commit (see below).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS, CONFD_ERR_NOSESSION, CONFD_ERR_EXTERNAL

```
int maapi_candidate_confirmed_commit_persistent(int sock, int timeoutsecs, const char *persist, const char *persist_id);
```

This function can be used to start or extend a persistent confirmed commit. The `persist` parameter sets the cookie for the persistent confirmed commit, while the `persist_id` gives the cookie for an already ongoing persistent confirmed commit. This gives the following possibilities:

`persist = "cookie",`
`persist_id = NULL`

Start a persistent confirmed commit with the cookie "cookie", or extend an already ongoing non-persistent confirmed commit and turn it into a persistent confirmed commit.

`persist = "newcookie",`
`persist_id = "oldcookie"`

Extend an ongoing persistent confirmed commit that uses the cookie "oldcookie" and change the cookie to "newcookie".

`persist = NULL,`
`persist_id = "cookie"`

Extend an ongoing persistent confirmed commit that uses the cookie "oldcookie" and turn it into a non-persistent confirmed commit.

<code>persist = NULL,</code>	Does the same as
<code>persist_id = NULL</code>	<code>maapi_candidate_confirmed_commit()</code> .

Typical usage is to start a persistent confirmed commit with `persist = "cookie"`, `persist_id = NULL`, and to extend it with `persist = "cookie"`, `persist_id = "cookie"`.

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit, but `persist_id` didn't give the right cookie for it.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_INUSE`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_EXTERNAL`

```
int maapi_candidate_confirmed_commit_info(int sock, int timeoutsecs,
const char *persist, const char *persist_id, const char *label, const
char *comment);
```

This function does the same as `maapi_candidate_confirmed_commit_persistent()`, but allows for setting the "Label" and/or "Comment" that is stored in the rollback file when the candidate is committed to running. To set only the "Label", give `comment` as `NULL`, and to set only the "Comment", give `label` as `NULL`. If both `label` and `comment` are `NULL`, the function does exactly the same as `maapi_candidate_confirmed_commit_persistent()`.



Note

To ensure that the "Label" and/or "Comment" are stored in the rollback file in all cases when doing a confirmed commit, they must be given both with the confirmed commit (using this function) and with the confirming commit (using `maapi_candidate_commit_info()`).

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit, but `persist_id` didn't give the right cookie for it.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_INUSE`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_EXTERNAL`

```
int maapi_candidate_commit_persistent(int sock, const char
*persist_id);
```

Confirm an ongoing persistent confirmed commit with the cookie given by `persist_id`. If `persist_id` is `NULL`, it does the same as `maapi_candidate_commit()`.

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit, but `persist_id` didn't give the right cookie for it.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_INUSE`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_EXTERNAL`

```
int maapi_candidate_commit_info(int sock, const char *persist_id, const
char *label, const char *comment);
```

This function does the same as `maapi_candidate_commit_persistent()`, but allows for setting the "Label" and/or "Comment" that is stored in the rollback file when the candidate is committed to running. To set only the "Label", give `comment` as `NULL`, and to set only the "Comment", give `label` as `NULL`. If both `label` and `comment` are `NULL`, the function does exactly the same as `maapi_candidate_commit_persistent()`.

**Note**

To ensure that the "Label" and/or "Comment" are stored in the rollback file in all cases when doing a confirmed commit, they must be given both with the confirmed commit (using `maapi_candidate_confirmed_commit_info()`) and with the confirming commit (using this function).

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit, but `persist_id` didn't give the right cookie for it.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_INUSE`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_EXTERNAL`

```
int maapi_candidate_abort_commit_persistent(int sock, const char
*persist_id);
```

Cancel an ongoing persistent confirmed commit with the cookie given by `persist_id`. (If `persist_id` is `NULL`, it does the same as `maapi_candidate_abort_commit()`.)

If `confd_errno` is `CONFD_ERR_NOEXISTS` it means that there is an ongoing persistent confirmed commit, but `persist_id` didn't give the right cookie for it.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_INUSE`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_EXTERNAL`

```
int maapi_candidate_reset(int sock);
```

This function copies running into candidate.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_INUSE`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_NOSESSION`

```
int maapi_confirmed_commit_in_progress(int sock);
```

Checks whether a confirmed commit is ongoing. Returns 1 if some user session currently has an ongoing confirmed commit operation and 0 if not.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int maapi_copy_running_to_startup(int sock);
```

This function copies running to startup.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_INUSE`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`

```
int maapi_is_running_modified(int sock);
```

Returns 1 if running has been modified since the last copy to startup, 0 if it has not been modified.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`

```
int maapi_is_candidate_modified(int sock);
```

Returns 1 if candidate has been modified, i.e if there are any outstanding non committed changes to the candidate, 0 if no changes are done

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

TRANSACTION CONTROL

```
int maapi_start_trans(int sock, enum confd_dbname name, enum  
confd_trans_mode readwrite);
```

The main purpose of MAAPI is to provide read and write access into the NCS transaction manager. Regardless of whether data is kept in CDB or in some (or several) external data bases, the same API is used to access data. ConfD acts as a mediator and multiplexes the different commands to the code which is responsible for each individual data node.

This function creates a new transaction towards the data store specified by *name*, which can be one of CONFD_CANDIDATE, CONFD_OPERATIONAL, CONFD_RUNNING, or CONFD_STARTUP (however updating the startup data store is better done via `maapi_copy_running_to_startup()`). The *readwrite* parameter can be either CONFD_READ, to start a readonly transaction, or CONFD_READ_WRITE, to start a read-write transaction.

A readonly transaction will incur less resource usage, thus if no writes will be done (e.g. the purpose of the transaction is only to read operational data), it is best to use CONFD_READ. There are also some cases where starting a read-write transaction is not allowed, e.g. if we start a transaction towards the running data store and `/confdConfig/datastores/running/access` is set to "writable-through-candidate" in `confd.conf`, or if ConfD is running in HA slave mode.

If start of the transaction is successful, the function returns a new transaction handle, a non-negative integer *thandle* which must be used as a parameter in all API functions which manipulate the transaction.

We will drive this transaction forward through the different states a ConfD transaction goes through. See the ascii arts in [confd_lib_dp\(3\)](#) for a picture of these states. If an external database is used, and it has registered callback functions for the different transaction states, those callbacks will be called when we in MAAPI invoke the different MAAPI transaction manipulation functions. For example when we call `maapi_start_trans()` the `init()` callback will be invoked in all external databases. (However ConfD may delay the actual invocation of `init()` as an optimization, see [confd_lib_dp\(3\)](#).) If data is kept in CDB, ConfD will handle everything internally.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_TOOMANYTRANS, CONFD_ERR_BADSTATE, CONFD_ERR_NOT_WRITABLE

```
int maapi_start_trans2(int sock, enum confd_dbname name, enum  
confd_trans_mode readwrite, int usid);
```

If we want to start new transactions inside actions, we can use this function to execute the new transaction within the existing user session. It is equivalent to calling `maapi_set_user_session()` and then `maapi_start_trans()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_TOOMANYTRANS, CONFD_ERR_BADSTATE, CONFD_ERR_NOT_WRITABLE

```
int maapi_start_trans_flags(int sock, enum confd_dbname name, enum  
confd_trans_mode readwrite, int usid, int flags);
```

This function makes it possible to set the flags that can otherwise be used with `maapi_set_flags()` already when starting a transaction, as well as setting the MAAPI_FLAG_HIDE_INACTIVE and

MAAPI_FLAG_DELAYED_WHEN flags that can only be used with `maapi_start_trans_flags()`. See the description of `maapi_set_flags()` for the available flags. It also incorporates the functionality of `maapi_start_trans()` and `maapi_start_trans2()` with respect to user sessions: If `usid` is 0, the transaction will be started within the user session associated with the MAAPI socket (like `maapi_start_trans()`), otherwise it will be started within the user session given by `usid` (like `maapi_start_trans2()`).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_TOOMANYTRANS, CONFD_ERR_BADSTATE, CONFD_ERR_NOT_WRITABLE

```
int maapi_start_trans_flags2(int sock, enum confd_dbname dbname, enum confd_trans_mode readwrite, int usid, int flags, const char *vendor, const char *product, const char *version, const char *client_id);
```

This function does the same as `maapi_start_trans_flags()` but allows additional information about the transaction to be passed to NCS. Calling `maapi_start_trans_flags()` is equivalent to calling `maapi_start_trans_flags2()` with `vendor`, `product` and `version` set to NULL, and `client_id` set to `__MAAPI_CLIENT_ID__`. The `__MAAPI_CLIENT_ID__` macro (defined in `confd_maapi.h`) will expand to a string representation of `__FILE__:__LINE__`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_TOOMANYTRANS, CONFD_ERR_BADSTATE, CONFD_ERR_NOT_WRITABLE

```
int maapi_start_trans_in_trans(int sock, enum confd_trans_mode readwrite, int usid, int thandle);
```

This function makes it possible to start a transaction with another transaction as backend, instead of an actual data store. This can be useful if we want to make a set of related changes, and then either apply or discard them all based on some criterion, while other changes remain unaffected. The `thandle` identifies the backend transaction to use. If `usid` is 0, the transaction will be started within the user session associated with the MAAPI socket, otherwise it will be started within the user session given by `usid`. If we call `maapi_apply_trans()` for this "transaction in a transaction", the changes (if any) will be applied to the backend transaction. To discard the changes, call `maapi_finish_trans()` without calling `maapi_apply_trans()` first.

The changes in this transaction can be validated by calling `maapi_validate_trans()` with a non-zero value for `forcevalidation`, but calling `maapi_apply_trans()` will not do any validation - in either case, the resulting configuration will be validated when the backend transaction is committed to the running data store. Note though that unlike the case with a transaction directly towards a data store, no transaction lock is taken on the underlying data store when doing validation of this type of transaction - thus it is possible for the contents of the data store to change (due to commit of another transaction) during the validation.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_TOOMANYTRANS, CONFD_ERR_BADSTATE

```
int maapi_finish_trans(int sock, int thandle);
```

This will finish the transaction. If the transaction is implemented by an external database, this will invoke the `finish()` callback.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

The error CONFD_ERR_NOEXISTS is set for all API functions which use a `thandle`, the return value from `maapi_start_trans()`, whenever no transaction is started.

```
int maapi_validate_trans(int sock, int thandle, int unlock, int
forcevalidation);
```

This function validates all data written in the transaction. This includes all data model constraints and all defined semantic validation in C, i.e. user programs that have registered functions under validation points. (See the [Semantic Validation chapter in the User Guide](#).)

If this function returns `CONFD_ERR`, the transaction is open for further editing. There are two special `confd_errno` values which are of particular interest here.

<code>CONFD_ERR_EXTERNAL</code>	this means that an external validation program in C returns <code>CONFD_ERR</code> i.e. that the semantic validation failed. The reason for the failure can be found in <code>confd_lasterr()</code>
<code>CONFD_ERR_VALIDATION_WARNING</code>	means that an external semantic validation program in C returned <code>CONFD_VALIDATION_WARN</code> . The string <code>confd_lasterr()</code> is organized as a series of NUL terminated strings as in <code>keypath1</code> , <code>reason1</code> , <code>keypath2</code> , <code>reason2</code> ... where the sequence is terminated with an additional NUL

If `unlock` is 1, the transaction is open for further editing even if validation succeeds. If `unlock` is 0 and the function returns `CONFD_OK`, the next function to be called **MUST** be `maapi_prepare_trans()` or `maapi_finish_trans()`.

`unlock = 1` can be used to implement a 'validate' command which can be given in the middle of an editing session. The first thing that happens is that a lock is set. If `unlock == 1`, the lock is released on success. The lock is always released on failure.

The *forcevalidation* parameter should normally be 0. It has no effect for a transaction towards the running or startup data stores, validation is always performed. For a transaction towards the candidate data store, validation will not be done unless *forcevalidation* is non-zero. Avoiding this validation is preferable if we are going to commit the candidate to running (e.g. with `maapi_candidate_commit()`), since otherwise the validation will be done twice. However if we are implementing a 'validate' command, we should give a non-zero value for *forcevalidation*.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_NOTSET`, `CONFD_ERR_NON_UNIQUE`, `CONFD_ERR_BAD_KEYREF`, `CONFD_ERR_TOO_FEW_ELEMS`, `CONFD_ERR_TOO_MANY_ELEMS`, `CONFD_ERR_UNSET_CHOICE`, `CONFD_ERR_MUST_FAILED`, `CONFD_ERR_MISSING_INSTANCE`, `CONFD_ERR_INVALID_INSTANCE`, `CONFD_ERR_INUSE`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_BADSTATE`

```
int maapi_prepare_trans(int sock, int thandle);
```

This function must be called as first part of two-phase commit. After this function has been called `maapi_commit_trans()` or `maapi_abort_trans()` must be called.

It will invoke the prepare callback in all participants in the transaction. If all participants reply with `CONFD_OK`, the second phase of the two-phase commit procedure is commenced.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_EXTERNAL`, `CONFD_ERR_NOTSET`, `CONFD_ERR_BADSTATE`, `CONFD_ERR_INUSE`

```
int maapi_commit_trans(int sock, int thandle);
```

```
int maapi_abort_trans(int sock, int thandle);
```

Finally at the last stage, either commit or abort must be called. A call to one of these functions must also eventually be followed by a call to `maapi_finish_trans()` which will terminate the transaction.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_NOEXISTS, CONFID_ERR_EXTERNAL, CONFID_ERR_BADSTATE

```
int maapi_apply_trans(int sock, int thandle, int keepopen);
```

Invoking the above transaction functions in exactly the right order can be a bit complicated. The right order to invoke the functions is `maapi_validate_trans()`, `maapi_prepare_trans()`, `maapi_commit_trans()` (or `maapi_abort_trans()`). Usually we do not require this fine grained control over the two-phase commit protocol. It is easier to use `maapi_apply_trans()` which validates, prepares and eventually commits or aborts.

A call to `maapi_apply_trans()` must also eventually be followed by a call to `maapi_finish_trans()` which will terminate the transaction.



Note

For a readonly transaction, i.e. one started with `readwrite == CONFID_READ`, or for a read-write transaction where we haven't actually done any writes, we do not need to call any of the validate/prepare/commit/abort or apply functions, since there is nothing for them to do. Calling `maapi_finish_trans()` to terminate the transaction is sufficient.

The parameter `keepopen` can optionally be set to 1, then the changes to the transaction are not discarded if validation fails. This feature is typically used by management applications that wish to present the validation errors to an operator and allow the operator to fix the validation errors and then later retry the apply sequence.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_NOEXISTS, CONFID_ERR_NOTSET, CONFID_ERR_NON_UNIQUE, CONFID_ERR_BAD_KEYREF, CONFID_ERR_TOO_FEW_ELEMS, CONFID_ERR_TOO_MANY_ELEMS, CONFID_ERR_UNSET_CHOICE, CONFID_ERR_MUST_FAILED, CONFID_ERR_MISSING_INSTANCE, CONFID_ERR_INVALID_INSTANCE, CONFID_ERR_INUSE, CONFID_ERR_BADTYPE, CONFID_ERR_EXTERNAL, CONFID_ERR_BADSTATE

READ/WRITE FUNCTIONS

```
int maapi_set_namespace(int sock, int thandle, int hashed_ns);
```

If we want to read or write data where the toplevel element name is not unique, we must indicate which namespace we are going to use. It is possible to change the namespace several times during a transaction.

The `hashed_ns` integer is the integer which is defined for the namespace in the .h file which is generated by the 'confdc' compiler. It is also possible to indicate which namespace to use through the namespace prefix when we read and write data. Thus the path `/foo:bar/baz` will get us `/bar/baz` in the namespace with prefix "foo" regardless of what the "set" namespace is. And if there is only one toplevel element called "bar" across all namespaces, we can use `/bar/baz` without the prefix and without calling `maapi_set_namespace()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

```
int maapi_cd(int sock, int thandle, const char *fmt, ...);
```

This function mimics the behavior of the UNIX "cd" command. It changes our working position in the data tree. If we are worried about performance, it is more efficient to invoke `maapi_cd()` to some position in the tree and there perform a series of operations using relative paths than it is to perform the equivalent series of operations using absolute paths. Note that this function can not be used as an existence test.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS

```
int maapi_pushd(int sock, int thandle, const char *fmt, ...);
```

Behaves like `maapi_cd()` with the exception that we can subsequently call `maapi_popd()` and returns to the previous position in the data tree.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOSTACK, CONFD_ERR_NOEXISTS

```
int maapi_popd(int sock, int thandle);
```

Pops the top position of the directory stack and changes directory.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOSTACK, CONFD_ERR_NOEXISTS

```
int maapi_getcwd(int sock, int thandle, size_t strsz, char *curdir);
```

Returns the current position as previously set by `maapi_cd()`, `maapi_pushd()`, or `maapi_popd()` as a string. Note that what is returned is a pretty-printed version of the internal representation of the current position, it will be the shortest unique way to print the path but it might not exactly match the string given to `maapi_cd()`. The buffer in `*curdir` will be NULL terminated, and no more characters than `strsz-1` will be written to it.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

```
int maapi_getcwd_kpath(int sock, int thandle, confd_hkeypath_t **kp);
```

Returns the current position like `maapi_getcwd()`, but as a pointer to a hashed keypath instead of as a string. The `hkeypath` is dynamically allocated, and may further contain dynamically allocated elements. The caller must free the allocated memory, easiest done by calling `confd_free_hkeypath()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

```
int maapi_exists(int sock, int thandle, const char *fmt, ...);
```

Boolean function which return 1 if the path refers to an existing node in the data tree, 0 if it does not.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS, CONFD_ERR_ACCESS_DENIED

```
int maapi_num_instances(int sock, int thandle, const char *fmt, ...);
```

Returns the number of entries for a list in the data tree.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION,
CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS, CONFD_ERR_ACCESS_DENIED

```
int maapi_get_elem(int sock, int thandle, confd_value_t *v, const char  
*fmt, ...);
```

This function reads a value from the path in *fmt* and writes the result into the result parameter *confd_value_t*. The path must lead to a leaf node in the data tree. Note that for the C_BUF, C_BINARY, C_LIST, C_OBJECTREF, C_OID, C_QNAME, C_HEXSTR, and C_BITBIG *confd_value_t* types, the buffer(s) pointed to are allocated using `malloc(3)` - it is up to the user of this interface to free them using `confd_free_value()`.

The `maapi` interface also contains a long list of access functions that accompany the `maapi_get_elem()` function which is a general access function that returns a *confd_value_t*. The accompanying functions all have the format `maapi_get_<type>_elem()` where <type> is one of the actual C types a *confd_value_t* can have. For example the function:

```
maapi_get_int64_elem(int sock, int thandle, int64_t *rval,  
                    const char *fmt, ...);
```

is used to read a signed 64 bit integer. It fills in the provided *int64_t* parameter. This corresponds to the YANG datatype `int64`, see [confd_types\(3\)](#). Similar access functions are provided for all the different builtin types.

One access function that needs additional explaining is the `maapi_get_str_elem()`. This function copies at most *n-1* characters into a user provided buffer, and terminates the string with a NUL character. If the buffer is not sufficiently large CONFD_ERR is returned, and `confd_errno` is set to CONFD_ERR_PROTOUSAGE. Note it is always possible to use `maapi_get_elem()` to get hold of the *confd_value_t*, which in the case of a string buffer contains the length.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH,
CONFD_ERR_NOEXISTS, CONFD_ERR_ACCESS_DENIED, CONFD_ERR_PROTOUSAGE,
CONFD_ERR_BADTYPE

```
int maapi_get_int8_elem(int sock, int thandle, int8_t *rval, const char  
*fmt, ...);
```

```
int maapi_get_int16_elem(int sock, int thandle, int16_t *rval, const  
char *fmt, ...);
```

```
int maapi_get_int32_elem(int sock, int thandle, int32_t *rval, const  
char *fmt, ...);
```

```
int maapi_get_int64_elem(int sock, int thandle, int64_t *rval, const  
char *fmt, ...);
```

```
int maapi_get_u_int8_elem(int sock, int thandle, u_int8_t *rval, const  
char *fmt, ...);
```

```
int maapi_get_u_int16_elem(int sock, int thandle, u_int16_t *rval,  
const char *fmt, ...);
```

```
int maapi_get_u_int32_elem(int sock, int thandle, u_int32_t *rval,  
const char *fmt, ...);
```

```

int maapi_get_u_int64_elem(int sock, int thandle, u_int64_t *rval,
const char *fmt, ...);

int maapi_get_ipv4_elem(int sock, int thandle, struct in_addr *rval,
const char *fmt, ...);

int maapi_get_ipv6_elem(int sock, int thandle, struct in6_addr *rval,
const char *fmt, ...);

int maapi_get_double_elem(int sock, int thandle, double *rval, const
char *fmt, ...);

int maapi_get_bool_elem(int sock, int thandle, int *rval, const char
*fmt, ...);

int maapi_get_datetime_elem(int sock, int thandle, struct
confd_datetime *rval, const char *fmt, ...);

int maapi_get_date_elem(int sock, int thandle, struct confd_date *rval,
const char *fmt, ...);

int maapi_get_gyearmonth_elem(int sock, int thandle, struct
confd_gYearMonth *rval, const char *fmt, ...);

int maapi_get_gyear_elem(int sock, int thandle, struct confd_gYear
*rval, const char *fmt, ...);

int maapi_get_time_elem(int sock, int thandle, struct confd_time *rval,
const char *fmt, ...);

int maapi_get_gday_elem(int sock, int thandle, struct confd_gDay *rval,
const char *fmt, ...);

int maapi_get_gmonthday_elem(int sock, int thandle, struct
confd_gMonthDay *rval, const char *fmt, ...);

int maapi_get_month_elem(int sock, int thandle, struct confd_gMonth
*rval, const char *fmt, ...);

int maapi_get_duration_elem(int sock, int thandle, struct
confd_duration *rval, const char *fmt, ...);

int maapi_get_enum_value_elem(int sock, int thandle, int32_t *rval,
const char *fmt, ...);

int maapi_get_bit32_elem(int sock, int th, int32_t *rval, const char
*fmt, ...);

int maapi_get_bit64_elem(int sock, int th, int64_t *rval, const char
*fmt, ...);

int maapi_get_oid_elem(int sock, int th, struct confd_snmp_oid **rval,
const char *fmt, ...);

int maapi_get_buf_elem(int sock, int thandle, unsigned char **rval, int
*bufsiz, const char *fmt, ...);

```

```

int maapi_get_str_elem(int sock, int th, char *buf, int n, const char
*fmt, ...);

int maapi_get_binary_elem(int sock, int thandle, unsigned char **rval,
int *bufsiz, const char *fmt, ...);

int maapi_get_qname_elem(int sock, int thandle, unsigned char
**prefix, int *prefixsz, unsigned char **name, int *namesz, const char
*fmt, ...);

int maapi_get_list_elem(int sock, int th, confd_value_t **values, int
*n, const char *fmt, ...);

int maapi_get_ipv4prefix_elem(int sock, int thandle, struct
confd_ipv4_prefix *rval, const char *fmt, ...);

int maapi_get_ipv6prefix_elem(int sock, int thandle, struct
confd_ipv6_prefix *rval, const char *fmt, ...);

```

Similar to the CDB API, MAAPI also includes typesafe variants for all the builtin types. See [confd_types\(3\)](#).

```

int maapi_vget_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, va_list args);

```

This function does the same as `maapi_get_elem()`, but takes a single `va_list` argument instead of a variable number of arguments - i.e. similar to `vprintf()`. Corresponding `va_list` variants exist for all the functions that take a path as a variable number of arguments.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`,
`CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_PROTOUSAGE`,
`CONFD_ERR_BADTYPE`

```

int maapi_init_cursor(int sock, int thandle, struct maapi_cursor *mc,
const char *fmt, ...);

```

Whenever we wish to iterate over the entries in a list in the data tree, we must first initialize a cursor. The cursor is subsequently used in a while loop.

For example if we have:

```

container servers {
  list server {
    key name;
    max-elements 64;
    leaf name {
      type string;
    }
    leaf ip {
      type inet:ip-address;
    }
    leaf port {
      type inet:port-number;
      mandatory true;
    }
  }
}

```

We can have the following C code which iterates over all server entries.

```
struct maapi_cursor mc;

maapi_init_cursor(sock, th, &mc, "/servers/server");
maapi_get_next(&mc);
while (mc.n != 0) {
    ... do something
    maapi_get_next(&mc);
}
maapi_destroy_cursor(&mc);
```

When a `tailf:secondary-index` statement is used in the data model (see [tailf_yang_extensions\(5\)](#)), we can set the `secondary_index` element of the struct `maapi_cursor` to indicate the name of a chosen secondary index - this must be done after the call to `maapi_init_cursor()` (which sets `secondary_index` to `NULL`) and before any call to `maapi_get_next()`. In this case, `secondary_index` must point to a NUL-terminated string that is valid throughout the iteration.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`

```
int maapi_get_next(struct maapi_cursor *mc);
```

Iterates and gets the keys for the next entry in a list. The key(s) can be used to retrieve further data. The key(s) are stored as `confd_value_t` structures in an array inside the struct `maapi_cursor`. The array of keys will be deallocated by the library.

For example to read the `port` leaf from an entry in the `server` list above, we would do:

```
....
maapi_init_cursor(sock, th, &mc, "/servers/server");
maapi_get_next(&mc);
while (mc.n != 0) {
    confd_value_t v;
    maapi_get_elem(sock, th, &v, "/servers/server{%x}/port", &mc.keys[0]);
    ....
    maapi_get_next(&mc);
}
```

The `'%x'` modifier (see the `PATHS` section in [confd_lib_cdb\(3\)](#)) is especially useful when working with a `maapi` cursor. The example above assumes that we know that the `/servers/server` list has exactly one key. But we can alternatively write `maapi_get_elem(sock, th, &v, "/servers/server{%x}/port", mc.n, mc.keys);` - which works regardless of the number of keys that the list has.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`

```
int maapi_find_next(struct maapi_cursor *mc, enum confd_find_next_type
type, confd_value_t *inkeys, int n_inkeys);
```

Update the cursor `mc` with the key(s) for the list entry designated by the `type` and `inkeys` parameters. This function may be used to start a traversal from an arbitrary entry in a list. Keys for subsequent entries may be retrieved with the `maapi_get_next()` function.

The `inkeys` array is populated with `n_inkeys` values that designate the starting point in the list. Normally the array is populated with key values for the list, but if the `secondary_index` element of the cursor has been set, the array must instead be populated with values for the corresponding secondary index-leaves. The `type` can have one of two values:

CONFID_FIND_NEXT

The keys for the first list entry *after* the one indicated by the *inkeys* array are requested. The *inkeys* array does not have to correspond to an actual existing list entry. Furthermore the number of values provided in the array (*n_inkeys*) may be fewer than the number of keys (or number of index-leafs for a secondary-index) in the data model, possibly even zero. This indicates that only the first *n_inkeys* values are provided, and the remaining ones should be taken to have a value "earlier" than the value for any existing list entry.

CONFID_FIND_SAME_OR_NEXT

If the values in the *inkeys* array completely identify an actual existing list entry, the keys for this entry are requested. Otherwise the same logic as described for CONFID_FIND_NEXT is used.

The following example will traverse the *server* list starting with the first entry (if any) that has a key value that is after "smtp" in the list order:

```
....
confd_value_t inkeys[1];

maapi_init_cursor(sock, th, &mc, "/servers/server");
CONFID_SET_STR(&inkeys[0], "smtp");

maapi_find_next(&mc, CONFID_FIND_NEXT, inkeys, 1);
while (mc.n != 0) {
    confd_value_t v;
    maapi_get_elem(sock, th, &v, "/servers/server{%x}/port", &mc.keys[0]);
    ....
    maapi_get_next(&mc);
}
```

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION,
CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_ACCESS_DENIED

```
void maapi_destroy_cursor(struct maapi_cursor *mc);
```

Deallocates memory which is associated with the cursor.

```
int maapi_set_elem(int sock, int thandle, confd_value_t *v, const char
*fmt, ...);
```

```
int maapi_set_elem2(int sock, int thandle, const char *strval, const
char *fmt, ...);
```

We have two different functions to set values. One where the value is a string and one where the value to set is a *confd_value_t*. The string version is useful when we have implemented a management agent where the user enters values as strings. The version with *confd_value_t* is useful when we are setting values which we have just read.

Another note which might effect users is that if the type we are writing is any of the encrypt or hash types, the *maapi_set_elem2()* will perform the asymmetric conversion of values whereas the *maapi_set_elem()* will not. See [confd_types\(3\)](#), the types *tailf:md5-digest-string*, *tailf:des3-cbc-encrypted-string*, and *tailf:aes-cfb-128-encrypted-string*.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION,
CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_BADTYPE,
CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_INUSE

```
int maapi_vset_elem(int sock, int thandle, confd_value_t *v, const char *fmt, va_list args);
```

This function does the same as `maapi_set_elem()`, but takes a single `va_list` argument instead of a variable number of arguments - i.e. similar to `vprintf()`. Corresponding `va_list` variants exist for all the functions that take a path as a variable number of arguments.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_BADTYPE, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_INUSE

```
int maapi_create(int sock, int thandle, const char *fmt, ...);
```

Create a new list entry, a presence container, or a leaf of type `empty` in the data tree. For example:
`maapi_create(sock, th, "/servers/server{www}")`;

If we are creating a new server entry as above, we must also populate all other data nodes below, which do not have a default value in the data model. Thus we must also do e.g.:

```
maapi_set_elem2(sock, th, "80", "/servers/server{www}/port");
```

before we try to commit the data.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_BADTYPE, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_NOTCREATABLE, CONFID_ERR_INUSE, CONFID_ERR_ALREADY_EXISTS

```
int maapi_delete(int sock, int thandle, const char *fmt, ...);
```

Delete an existing list entry, a presence container, or an optional leaf and all its children (if any) from the data tree.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_BADTYPE, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_NOTDELETABLE, CONFID_ERR_INUSE

```
int maapi_get_object(int sock, int thandle, confd_value_t *values, int n, const char *fmt, ...);
```

This function reads at most `n` values from the list entry or container specified by the path, and places them in the `values` array, which is provided by the caller. The array is populated according to the specification of the Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

On success, the function returns the actual number of elements needed. I.e. if the return value is bigger than `n`, only the values for the first `n` elements are in the array, and the remaining values have been discarded. Note that given the specification of the array contents, there is always a fixed upper bound on the number of actual elements, and if there are no presence sub-containers, the number is constant. See the description of `cdb_get_object()` in [confd_lib_cdb\(3\)](#) for usage examples - they apply to `maapi_get_object()` as well.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_ACCESS_DENIED

```
int maapi_get_objects(struct maapi_cursor *mc, confd_value_t *values, int n, int *nobj);
```

Similar to `maapi_get_object()`, but reads multiple list entries based on a struct `maapi_cursor`. At most `n` values from each of at most `*nobj` list entries, starting at the entry after the one given by `*mc`, are read and placed in the `values` array. The cursor must have been initialized with `maapi_init_cursor()` at some point before the call, but in principle it is possible to mix calls to `maapi_get_next()` and `maapi_get_objects()` using the same cursor.

The array must be at least `n * *nobj` elements long, and the values for entry `i` start at element `array[i * n]` (i.e. the first entry read starts at `array[0]`, the second at `array[n]`, and so on). On success, the highest actual number of values in any of the entries read is returned. If we attempt to read more entries than actually exist (i.e. if there are less than `*nobj` entries after the entry indicated by `*mc`), `*nobj` is updated with the actual number (possibly 0) of entries read. In this case the `n` element of the cursor is set to 0 as for `maapi_get_next()`. Example - read the data for all entries in the "server" list above, in chunks of 10:

```
#define VALUES_PER_ENTRY 3
#define ENTRIES_PER_REQUEST 10

struct maapi_cursor mc;
confd_value_t v[ENTRIES_PER_REQUEST*VALUES_PER_ENTRY];
int nobj, ret, i;

maapi_init_cursor(sock, th, &mc, "/servers/server");
do {
    nobj = ENTRIES_PER_REQUEST;
    ret = maapi_get_objects(&mc, v, VALUES_PER_ENTRY, &nobj);
    if (ret >= 0) {
        for (i = 0; i < nobj; i++) {
            ... process entry starting at v[i*VALUES_PER_ENTRY] ...
        }
    } else {
        ... handle error ...
    }
} while (ret >= 0 && mc.n != 0);
maapi_destroy_cursor(&mc);
```

See also the description of `cdb_get_object()` in [confd_lib_cdb\(3\)](#) for examples on how to use loaded schema information to avoid "hardwiring" constants like `VALUES_PER_ENTRY` above, and the relative position of individual leaf values in the value array.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`

```
int maapi_get_values(int sock, int thandle, confd_tag_value_t *values,
int n, const char *fmt, ...);
```

Read an arbitrary set of sub-elements of a container or list entry. The `values` array must be pre-populated with `n` values based on the specification of the *Tagged Value Array* format in the *XML STRUCTURES* section of the [confd_types\(3\)](#) manual page, where the `confd_value_t` value element is given as follows:

- `C_NOEXISTS` means that the value should be read from the transaction and stored in the array.
- `C_PTR` also means that the value should be read from the transaction, but instead gives the expected type and a pointer to the type-specific variable where the value should be stored. Thus this gives a functionality similar to the type safe `maapi_get_xxx_elem()` functions.
- `C_XMLBEGIN` and `C_XMLEND` are used as per the specification.
- Keys to select list entries can be given with their values.

**Note**

When we use `C_PTR`, we need to take special care to free any allocated memory. When we use `C_NOEXISTS` and the value is stored in the array, we can just use `confd_free_value()` regardless of the type, since the `confd_value_t` has the type information. But with `C_PTR`, only the actual value is stored in the pointed-to variable, just as for `maapi_get_buf_elem()`, `maapi_get_binary_elem()`, etc, and we need to free the memory specifically allocated for the types listed in the description of `maapi_get_elem()` above. The details of how to do this are not given for the `maapi_get_xxx_elem()` functions here, but it is the same as for the corresponding `cdb_get_xxx()` functions, see [confd_lib_cdb\(3\)](#).

All elements have the same position in the array after the call, in order to simplify extraction of the values - this means that optional elements that were requested but didn't exist will have `C_NOEXISTS` rather than being omitted from the array. However requesting a list entry that doesn't exist is an error. Note that when using `C_PTR`, the only indication of a non-existing value is that the destination variable has not been modified - it's up to the application to set it to some "impossible" value before the call when optional leafs are read.

**Note**

Selection of a list entry by its "instance integer", which can be done with `cdb_get_values()` by using `C_CDBBEGIN`, can *not* be done with `maapi_get_values()`

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`

```
int maapi_set_object(int sock, int thandle, const confd_value_t
*values, int n, const char *fmt, ...);
```

Set all leafs corresponding to the complete contents of a list entry or container, excluding for sub-lists. The *values* array must be populated with *n* values according to the specification of the Value Array format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page. Additionally, since operational data cannot be written, array elements corresponding to operational data leafs or containers must have the value `C_NOEXISTS`.

If the node specified by the path, or any sub-nodes that are specified as existing, do not exist before this call, they will be created, otherwise the existing values will be updated. Nodes that can be deleted and are specified as not existing in the array, i.e. with value `C_NOEXISTS`, will be deleted if they existed before the call.

For a list entry, since the key values must be present in the array, it is not required that the key values are included in the path given by *fmt*. If the key values *are* included in the path, the key values in the array are ignored.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_INUSE`

```
int maapi_set_values(int sock, int thandle, const confd_tag_value_t
*values, int n, const char *fmt, ...);
```

Set arbitrary sub-elements of a container or list entry. The *values* array must be populated with *n* values according to the specification of the *Tagged Value Array* format in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page.

If the container or list entry itself, or any sub-elements that are specified as existing, do not exist before this call, they will be created, otherwise the existing values will be updated. Both mandatory and optional elements may be omitted from the array, and all omitted elements are left unchanged. To actually delete a non-mandatory leaf or presence container as described for `maapi_set_object()`, it may (as an extension of the format) be specified as `C_NOEXISTS` instead of being omitted.

For a list entry, the key values can be specified either in the path or via key elements in the array - if the values are in the path, the key elements can be omitted from the array. For sub-lists present in the array, the key elements must of course always also be present though, immediately following the `C_XMLBEGIN` element and in the order defined by the data model. It is also possible to delete a list entry by using a `C_XMLBEGINDEL` element, followed by the keys in data model order, followed by a `C_XMLEND` element.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_INUSE`

```
int maapi_get_case(int sock, int thandle, const char *choice,
confd_value_t *rcase, const char *fmt, ...);
```

When we use the YANG choice statement in the data model, this function can be used to find the currently selected case, avoiding useless `maapi_get_elem()` etc requests for nodes that belong to other cases. The `fmt, ...` arguments give the path to the list entry or container where the choice is defined, and `choice` is the name of the choice. The case value is returned to the `confd_value_t` that `rcase` points to, as type `C_XMLTAG` - i.e. we can use the `CONFD_GET_XMLTAG()` macro to retrieve the hashed tag value.

If we have "nested" choices, i.e. multiple levels of choice statements without intervening container or list statements in the data model, the `choice` argument must give a '/'-separated path with alternating choice and case names, from the data node given by the `fmt, ...` arguments to the specific choice that the request pertains to.

For a choice without a mandatory `true` statement where no case is currently selected, the function will fail with `CONFD_ERR_NOEXISTS` if the choice doesn't have a default case. If it has a default case, it will be returned unless the `MAAPI_FLAG_NO_DEFAULTS` flag is in effect (see `maapi_set_flags()` below) - if the flag is set, the value returned via `rcase` will have type `C_DEFAULT`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_ACCESS_DENIED`

```
int maapi_get_attr(int sock, int thandle, u_int32_t *attrs, int
num_attrs, confd_attr_value_t **attr_vals, int *num_vals, const char
*fmt, ...);
```

Retrieve attributes for a configuration node. These attributes are currently supported:

```
/* CONFD_ATTR_TAGS: value is C_LIST of C_BUF/C_STR */
#define CONFD_ATTR_TAGS 0x80000000
/* CONFD_ATTR_ANNOTATION: value is C_BUF/C_STR */
#define CONFD_ATTR_ANNOTATION 0x80000001
/* CONFD_ATTR_INACTIVE: value is C_BOOL 1 (i.e. "true") */
#define CONFD_ATTR_INACTIVE 0x00000000
```

The `attrs` parameter is an array of attributes of length `num_attrs`, specifying the wanted attributes - if `num_attrs` is 0, all attributes are retrieved. If no attributes are found, `*num_vals` is set to 0, otherwise

an array of `confd_attr_value_t` elements is allocated and populated, its address stored in `*attr_vals`, and `*num_vals` is set to the number of elements in the array. The `confd_attr_value_t` struct is defined as:

```
typedef struct confd_attr_value {
    u_int32_t attr;
    confd_value_t v;
} confd_attr_value_t;
```

If any attribute values are returned (`*num_vals > 0`), the caller must free the allocated memory by calling `confd_free_value()` for each of the `confd_value_t` elements, and `free(3)` for the `*attr_vals` array itself.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_NOSESSION`, `CONFID_ERR_BADPATH`, `CONFID_ERR_NOEXISTS`, `CONFID_ERR_ACCESS_DENIED`, `CONFID_ERR_UNAVAILABLE`

```
int maapi_set_attr(int sock, int thandle, u_int32_t attr, confd_value_t
*v, const char *fmt, ...);
```

Set an attribute for a configuration node. See `maapi_get_attrs()` above for the supported attributes. To delete an attribute, call the function with a value of type `C_NOEXISTS`.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_NOSESSION`, `CONFID_ERR_BADPATH`, `CONFID_ERR_BADTYPE`, `CONFID_ERR_NOEXISTS`, `CONFID_ERR_ACCESS_DENIED`, `CONFID_ERR_UNAVAILABLE`

```
int maapi_delete_all(int sock, int thandle, enum maapi_delete_how how);
```

This function can be used to delete "all" the configuration data within a transaction. The *how* argument specifies the extent of "all":

<code>MAAPI_DEL_SAFE</code>	Delete everything except namespaces that were exported to none (with <code>tailf:export none</code>). Toplevel nodes that cannot be deleted due to AAA rules are silently left in place, but descendant nodes will still be deleted if the AAA rules allow it.
<code>MAAPI_DEL_EXPORTED</code>	Delete everything except namespaces that were exported to none (with <code>tailf:export none</code>). AAA rules are ignored, i.e. nodes are deleted even if the AAA rules don't allow it.
<code>MAAPI_DEL_ALL</code>	Delete everything. AAA rules are ignored.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_NOSESSION`, `CONFID_ERR_NOEXISTS`

```
int maapi_revert(int sock, int thandle);
```

This function removes all changes done to the transaction.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_NOSESSION`, `CONFID_ERR_NOEXISTS`

```
int maapi_set_flags(int sock, int thandle, int flags);
```

We can modify some aspects of the read/write session by calling this function - these values can be used for the *flags* argument (ORed together if more than one) with this function and/or with `maapi_start_trans_flags()`:

```
#define MAAPI_FLAG_HINT_BULK (1 << 0)
```

```
#define MAAPI_FLAG_NO_DEFAULTS    (1 << 1)
#define MAAPI_FLAG_CONFIG_ONLY    (1 << 2)
#define MAAPI_FLAG_HIDE_INACTIVE (1 << 3) /* maapi_start_trans_flags() only */
#define MAAPI_FLAG_DELAYED_WHEN  (1 << 6) /* maapi_start_trans_flags() only */
```

MAAPI_FLAG_HINT_BULK tells the ConfD backplane that we will be reading substantial amounts of data. This has the effect that the `get_object()` and `get_next_object()` callbacks (if available) are used towards external data providers when we call `maapi_get_elem()` etc and `maapi_get_next()`. The `maapi_get_object()` function always operates as if this flag was set.

MAAPI_FLAG_NO_DEFAULTS says that we want to be informed when we read leafs with default values that have not had a value set. This is indicated by the returned value being of type `C_DEFAULT` instead of the actual value. The default value for such leafs can be obtained from the `confd_cs_node` tree provided by the library (see [confd_types\(3\)](#)).

MAAPI_FLAG_CONFIG_ONLY will make the `maapi_get_xxx()` functions return config nodes only - if we attempt to read operational data, it will be treated as if the nodes did not exist. This is mainly useful in conjunction with `maapi_get_object()` and list entries or containers that have both config and operational data (the operational data nodes in the returned array will have the "value" `C_NOEXISTS`), but the other functions also obey the flag.

MAAPI_FLAG_HIDE_INACTIVE can only be used with `maapi_start_trans_flags()`, and only when starting a readonly transaction (parameter `readwrite == CONFD_READ`). It will hide configuration data that has the `CONFD_ATTR_INACTIVE` attribute set, i.e. it will appear as if that data does not exist.

MAAPI_FLAG_DELAYED_WHEN can also only be used with `maapi_start_trans_flags()`, but regardless of whether the flag is used or not, the "delayed when" mode can subsequently be changed with `maapi_set_delayed_when()`. The flag is only meaningful when starting a read-write transaction (parameter `readwrite == CONFD_READ_WRITE`), and will cause "delayed when" mode to be enabled from the beginning of the transaction. See the description of `maapi_set_delayed_when()` for information about the "delayed when" mode.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_NOEXISTS`

```
int maapi_set_delayed_when(int sock, int thandle, int on);
```

This function enables (*on* non-zero) or disables (*on* == 0) the "delayed when" mode of a transaction. When successful, it returns 1 or 0 as indication of whether "delayed when" was enabled or disabled before the call. See also the `MAAPI_FLAG_DELAYED_WHEN` flag for `maapi_start_trans_flags()`.

The YANG `when` statement makes its parent data definition statement conditional. This can be problematic in cases where we don't have control over the order of writing different data nodes. E.g. when loading configuration from a file, the data that will satisfy the `when` condition may occur after the data that the `when` applies to, making it impossible to actually write the latter data into the transaction - since the `when` isn't satisfied, the data nodes effectively do not exist in the schema.

This is addressed by the "delayed when" mode for a transaction. When "delayed when" is enabled, it is possible to write to data nodes even though they are conditional on a `when` that isn't satisfied. It has no effect on reading though - trying to read data that is conditional on an unsatisfied `when` will always result in `CONFD_ERR_NOEXISTS` or equivalent. When disabling "delayed when", any "delayed" `when` statements will take effect immediately - i.e. if the `when` isn't satisfied at that point, the conditional nodes and any data values for them will be deleted. If we don't explicitly disable "delayed when" by calling this function, it will be automatically disabled when the transaction enters the `VALIDATE` state (e.g. due to call of `maapi_apply_trans()`).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

```
int maapi_set_label(int sock, int thandle, const char *label);
```

Set the "Label" that is stored in the rollback file when a transaction towards running is committed. Setting the "Label" for transactions via candidate can be done when the candidate is committed to running, by using the `maapi_candidate_commit_info()` function. For a confirmed commit, the "Label" must also be given via the `maapi_candidate_confirmed_commit_info()` function.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

```
int maapi_set_comment(int sock, int thandle, const char *comment);
```

Set the "Comment" that is stored in the rollback file when a transaction towards running is committed. Setting the "Comment" for transactions via candidate can be done when the candidate is committed to running, by using the `maapi_candidate_commit_info()` function. For a confirmed commit, the "Comment" must also be given via the `maapi_candidate_confirmed_commit_info()` function.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_NOEXISTS

NCS SPECIFIC FUNCTIONS

The functions in this sections can only be used with NCS, and specifically the `maapi_shared_xxx()` functions must be used for NCS FASTMAP, i.e. in the service `create()` callback. Those functions maintain attributes that are necessary when multiple service instances modify the same data.

```
int maapi_shared_create(int sock, int thandle, int flags, const char *fmt, ...);
```

FASTMAP version of `maapi_create()`. Normally the *flags* parameter should be given as 0, but it is possible to suppress the creation of backpointer attributes by passing `MAAPI_SHARED_NO_BACKPOINTER` for *flags*.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS, CONFD_ERR_BADTYPE, CONFD_ERR_ACCESS_DENIED, CONFD_ERR_NOT_WRITABLE, CONFD_ERR_NOTCREATABLE, CONFD_ERR_INUSE

```
int maapi_shared_set_elem(int sock, int thandle, confd_value_t *v, int flags, const char *fmt, ...);
```

```
int maapi_shared_set_elem2(int sock, int thandle, const char *strval, int flags, const char *fmt, ...);
```

FASTMAP versions of `maapi_set_elem()` and `maapi_set_elem2()`. The *flags* parameter is currently unused and should be given as 0.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_BADPATH, CONFD_ERR_NOEXISTS, CONFD_ERR_BADTYPE, CONFD_ERR_ACCESS_DENIED, CONFD_ERR_NOT_WRITABLE, CONFD_ERR_INUSE

```
int maapi_shared_insert(int sock, int thandle, int flags, const char *fmt, ...);
```

FASTMAP version of `maapi_insert()`. Normally the *flags* parameter should be given as 0, but it is possible to suppress the creation of backpointer attributes by passing `MAAPI_SHARED_NO_BACKPOINTER` for *flags*.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_NOTDELETABLE`

```
int maapi_shared_set_values(int sock, int thandle, const
confd_tag_value_t *values, int n, int flags, const char *fmt, ...);
```

FASTMAP version of `maapi_set_values()`. Normally the *flags* parameter should be given as 0, but it is possible to suppress the creation of backpointer attributes by passing `MAAPI_SHARED_NO_BACKPOINTER` for *flags*.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_INUSE`

```
int maapi_shared_copy_tree(int sock, int thandle, int flags, const char
*from, const char *tofmt, ...);
```

FASTMAP version of `maapi_copy_tree()`. Normally the *flags* parameter should be given as 0, but it is possible to suppress the creation of backpointer attributes by passing `MAAPI_SHARED_NO_BACKPOINTER` for *flags*.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_BADPATH`

```
int maapi_ncs_apply_template(int sock, int thandle, char
*template_name, const struct ncs_name_value *variables, int
num_variables, int flags, const char *rootfmt, ...);
```

Apply a template that has been loaded into NCS. The *template_name* parameter gives the name of the template. The *variables* parameter is an *num_variables* long array of variables and names for substitution into the template. The struct `ncs_name_value` is defined as:

```
struct ncs_name_value {
    char *name;
    char *value;
};
```

The *flags* parameter is currently unused and should be given as 0.



Note

If this function is called under FASTMAP it will have the same behavior as the corresponding FASTMAP function `maapi_shared_ncs_apply_template()`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_XPATH`

```
int maapi_shared_ncs_apply_template(int sock, int thandle, char
*template_name, const struct ncs_name_value *variables, int
num_variables, int flags, const char *rootfmt, ...);
```

FASTMAP version of `maapi_ncs_apply_template()`. Normally the *flags* parameter should be given as 0.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE, CONFID_ERR_BADPATH, CONFID_ERR_NOEXISTS, CONFID_ERR_XPATH

```
int maapi_ncs_get_templates(int sock, char ***templates, int
*num_templates);
```

Retrieve a list of the templates currently loaded into NCS. On success, a pointer to an array of template names is stored in *templates* and the length of the array is stored in *num_templates*. The library allocates memory for the result, and the caller is responsible for freeing it. This can in all cases be done with code like this:

```
char **templates;
int num_templates, i;

if (maapi_ncs_get_templates(sock, &templates, &num_templates) == CONFID_OK) {
    ...
    for (i = 0; i < num_templates; i++) {
        free(templates[i]);
    }
    if (num_templates > 0) {
        free(templates);
    }
}
```

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS

MISCELLANEOUS FUNCTIONS

```
int maapi_delete_config(int sock, enum confd_dbname name);
```

This function empties a data store.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_EXTERNAL

```
int maapi_copy(int sock, int from_thandle, int to_thandle);
```

If we open two transactions from the same user session but towards different data stores, such as one transaction towards startup and one towards running, we can copy all data from one data store to the other with this function. This is a replace operation - any configuration that exists in the transaction given by *to_handle* but not in the one given by *from_handle* will be deleted from the *to_handle* transaction.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE

```
int maapi_copy_path(int sock, int from_thandle, int to_thandle, const
char *fmt, ...);
```

Similar to `maapi_copy()`, but does a replacing copy only of the subtree rooted at the path given by *fmt* and remaining arguments.

Errors: CONFID_ERR_MALLOC, CONFID_ERR_OS, CONFID_ERR_NOSESSION, CONFID_ERR_ACCESS_DENIED, CONFID_ERR_NOT_WRITABLE

```
int maapi_copy_tree(int sock, int thandle, const char *from, const char *tofmt, ...);
```

This function copies the entire configuration tree rooted at *from* to *tofmt*. List entries are created accordingly. If the destination already exists, *from* is copied on top of the destination. This function is typically used inside actions where we for example could use `maapi_copy_tree()` to copy a template configuration into a new list entry. The *from* path must be pre-formatted, e.g. using `confd_format_keypath()`, whereas the destination path is formatted by this function.



Note

The data models for the source and destination trees must match - i.e. they must either be identical, or the data model for the source tree must be a proper subset of the data model for the destination tree. This is always fulfilled when copying from one entry to another in a list, or if both source and destination tree have been defined via YANG `uses` statements referencing the same `grouping` definition. If a data model mismatch is detected, e.g. an existing data node in the source tree does not exist in the destination data model, or an existing leaf in the source tree has a value that is incompatible with the type of the leaf in the destination data model, `maapi_copy_tree()` will return `CONFD_ERR` with `confd_errno` set to `CONFD_ERR_BADPATH`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_BADPATH`

```
int maapi_insert(int sock, int thandle, const char *fmt, ...);
```

This function inserts a new entry in a list that uses the `tailf:indexed-view` statement. The key must be of type integer. If the inserted entry already exists, the existing and subsequent entries will be renumbered as needed, unless renumbering would require an entry to have a key value that is outside the range of the type for the key. In that case, the function returns `CONFD_ERR` with `confd_errno` set to `CONFD_ERR_BADTYPE`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_NOTDELETABLE`

```
int maapi_move(int sock, int thandle, confd_value_t* tokey, int n, const char *fmt, ...);
```

This function moves an existing list entry, i.e. renames the entry using the *tokey* parameter, which is an array containing *n* keys.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_NOT_WRITABLE`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_NOTMOVABLE`, `CONFD_ERR_ALREADY_EXISTS`

```
int maapi_move_ordered(int sock, int thandle, enum maapi_move_where where, confd_value_t* tokey, int n, const char *fmt, ...);
```

For a list with the YANG `ordered-by user` statement, this function can be used to change the order of entries, by moving one entry to a new position. When new entries in such a list are created with `maapi_create()`, they are always placed last in the list. The path given by *fmt* and the remaining arguments identifies the entry to move, and the new position is given by the *where* argument:

MAAPI_MOVE_FIRST Move the entry first in the list. The *tokey* and *n* arguments are ignored, and can be given as `NULL` and `0`.

MAAPI_MOVE_LAST Move the entry last in the list. The *tokey* and *n* arguments are ignored, and can be given as NULL and 0.

MAAPI_MOVE_BEFORE Move the entry to the position before the entry given by the *tokey* argument, which is an array of key values with length *n*.

MAAPI_MOVE_AFTER Move the entry to the position after the entry given by the *tokey* argument, which is an array of key values with length *n*.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION, CONFD_ERR_ACCESS_DENIED, CONFD_ERR_NOT_WRITABLE, CONFD_ERR_NOEXISTS, CONFD_ERR_NOTMOVABLE

```
int maapi_authenticate(int sock, const char *user, const char *pass,
char *groups[], int n);
```

If we are implementing a proprietary management agent with MAAPI API, the function `maapi_start_user_session()` requires the application to tell ConfD which groups the user are member of. ConfD itself has the capability to authenticate users. A MAAPI application can use `maapi_authenticate()` to let ConfD authenticate the user, as per the AAA configuration in `confd.conf`

If the authentication is successful, the function returns 1, and the `groups[]` array is populated with at most *n-1* NUL-terminated strings containing the group names, followed by a NULL pointer that indicates the end of the group list. The strings are dynamically allocated, and it is up to the caller to free the memory by calling `free(3)` for each string. If the function is used in a context where the group names are not needed, pass 1 for the *n* parameter.

If the authentication fails, the function returns 0, and `confd_lasterr()` (see [confd_lib_lib\(3\)](#)) will return a message describing the reason for the failure.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION

```
int maapi_authenticate2(int sock, const char *user, const char *pass,
const struct confd_ip *src_addr, int src_port, const char *context,
enum confd_proto prot, char *groups[], int n);
```

This function does the same thing as `maapi_authenticate()`, but allows for passing of the additional parameters `src_addr`, `src_port`, `context`, and `prot`, which otherwise are passed only to `maapi_start_user_session()/maapi_start_user_session2()`. These parameters are not used when ConfD performs the authentication, but they will be passed to an external authentication executable (see the [External authentication](#) section of the AAA chapter in the User Guide) if `/confdConfig/aaa/externalAuthentication/includeExtra` is set to "true" in `confd.conf`, see [confd.conf\(5\)](#). They will also be made available to the authentication callback that can be registered by an application (see [confd_lib_dp\(3\)](#)).

Errors: CONFD_ERR_PROTOUSAGE, CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOSESSION

```
int maapi_attach(int sock, int hashed_ns, struct confd_trans_ctx *ctx);
```

While ConfD is executing a transaction, we have a number of situations where we wish to invoke user C code which can interact in the transaction. One such situation is when we wish to write semantic validation code which is invoked in the validation phase of a ConfD transaction. This code needs to execute within the context of the executing transaction, it must thus have access to the "shadow" storage where all not-yet-committed data is kept.

This function attaches to a existing transaction. See the [Semantic Validation chapter in the User Guide](#) for example code.

Another situation where we wish to attach to the executing transaction is when we are using the notifications API and subscribe to notification of type CONFD_NOTIF_COMMIT_DIFF and wish to read the committed diffs from the transaction.

The *hashed_ns* parameter is basically just there to save a call to `maapi_set_namespace()`. We can call `maapi_set_namespace()` any number of times to change from the one we passed to `maapi_attach()`, and we can also give the namespace in prefix form in the *path* parameter to the read/write functions - see the `maapi_set_namespace()` description.

If we do not want to give a specific namespace when invoking `maapi_attach()`, we can give 0 for the *hashed_ns* parameter (-1 works too but is deprecated). We can still call the read/write functions as long as the toplevel element in the *path* is unique, but otherwise we must call `maapi_set_namespace()`, or use a prefix in the *path*.

```
int maapi_attach2(int sock, int hashed_ns, int usid, int thandle);
```

When we write proprietary CLI commands in C and we wish those CLI commands to be able to use MAAPI to read and write data inside the same transaction the CLI command was invoked in, we do not have an initialized transaction structure available. Then we must use this function. CLI commands get the *usid* passed in UNIX environment variable CONFD_MAAPI_USID and the *thandle* passed in environment variable CONFD_MAAPI_THANDLE. We also need to use this function when implementing such CLI commands via `action_command()` callbacks, see the [confd_lib_dp\(3\)](#) man page. In this case the *usid* is provided via `uinfo->usid` and the *thandle* via `uinfo->actx.thandle`. To use the user session id that is the owner of the transaction, set *usid* to 0. If the namespace does not matter set *hashed_ns* to 0, see `maapi_attach()`.

```
int maapi_attach_init(int sock, int *thandle);
```

This function is used to attach the MAAPI socket to the special transaction available in phase0 used for CDB initialization and upgrade. The function is also used if we need to modify CDB data during in-service data model upgrade (see the ["In-service Data Model Upgrade" chapter in the User Guide](#)). The transaction handle, which is used in subsequent calls to MAAPI, is filled in by the function upon successful return. See the [CDB chapter in the User Guide](#).

```
int maapi_detach(int sock, struct confd_trans_ctx *ctx);
```

Detaches an attached MAAPI socket. This function is typically called in the `stop()` callback in validation code. An attached MAAPI socket will be automatically detached when the ConfD transaction terminates. This function performs an explicit detach.

```
int maapi_detach2(int sock, int thandle);
```

Detaches an attached MAAPI socket when we do not have an initialized transaction structure available, see `maapi_attach2()` above. This is mainly useful in an `action_command()` callback.

```
int maapi_diff_iterate(int sock, int thandle, enum maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op op, confd_value_t *oldv, confd_value_t *newv, void *state), int flags, void *initstate);
```

This function can be called from an attached MAAPI session. The purpose of the function is to iterate through the transaction diff. It can typically be used in conjunction with the notification API when we subscribe to CONFD_NOTIF_COMMIT_DIFF events. It can also be used inside validation callbacks.

For all diffs in the transaction the supplied callback function `iter()` will be called. The `iter()` callback receives the `confd_hkeypath_t kp` which uniquely identifies which node in the data tree that is affected, the operation, and an optional value. The `op` parameter gives the modification as:

MOP_CREATED	The list entry, presence container, or leaf of type <code>empty</code> given by <code>kp</code> has been created.
MOP_DELETED	The list entry, presence container, or optional leaf given by <code>kp</code> has been deleted.
MOP_MODIFIED	A descendant of the list entry given by <code>kp</code> has been modified.
MOP_VALUE_SET	The value of the leaf given by <code>kp</code> has been set to <code>newv</code> . If the <code>MAAPI_FLAG_NO_DEFAULTS</code> flag has been set and the default value for the leaf has come into effect, <code>newv</code> will be of type <code>C_DEFAULT</code> instead of giving the default value.
MOP_MOVED_AFTER	The list entry given by <code>kp</code> , in an ordered-by user list, has been moved. If <code>newv</code> is <code>NULL</code> , the entry has been moved first in the list, otherwise it has been moved after the entry given by <code>newv</code> . In this case <code>newv</code> is a pointer to an array of key values identifying an entry in the list. The array is terminated with an element that has type <code>C_NOEXISTS</code> .
MOP_ATTR_SET	An attribute for the node given by <code>kp</code> has been modified (see the description of <code>maapi_get_attrs()</code> for the supported attributes). The <code>iter()</code> callback will only get this invocation when attributes are enabled in <code>confd.conf (/confdConfig/enableAttributes, see confd.conf(5))</code> and the flag <code>ITER_WANT_ATTR</code> has been passed to <code>maapi_diff_iterate()</code> . The <code>newv</code> parameter is a pointer to a 2-element array, where the first element is the attribute represented as a <code>confd_value_t</code> of type <code>C_UINT32</code> and the second element is the value the attribute was set to. If the attribute has been deleted, the second element is of type <code>C_NOEXISTS</code> .

The `oldv` parameter passed to `iter()` is always `NULL`.

If `iter()` returns `ITER_STOP`, no more iteration is done, and `CONFD_OK` is returned. If `iter()` returns `ITER_RECURSE` iteration continues with all children to the node. If `iter()` returns `ITER_CONTINUE` iteration ignores the children to the node (if any), and continues with the node's sibling. If, for some reason, the `iter()` function wants to return control to the caller of `maapi_diff_iterate()` before all the changes have been iterated over it can return `ITER_SUSPEND`. The caller then has to call `maapi_diff_iterate_resume()` to continue/finish the iteration.

The `flags` parameter is a bitmask with the following bits:

<code>ITER_WANT_ATTR</code>	Enable <code>MOP_ATTR_SET</code> invocations of the <code>iter()</code> function.
<code>ITER_WANT_P_CONTAINER</code>	Invoke <code>iter()</code> for modified presence-containers.

The `state` parameter can be used for any user supplied state (i.e. whatever is supplied as `init_state` is passed as `state` to `iter()` in each invocation).

The `iter()` invocations are not subjected to AAA checks, i.e. regardless of which path we have and which context was used to create the MAAPI socket, all changes are provided.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADSTATE`.

`CONFD_ERR_BADSTATE` is returned when we try to iterate on a transaction which is in the wrong state and not attached.

```
int maapi_keypath_diff_iterate(int sock, int thandle, enum
maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op op,
confd_value_t *oldv, confd_value_t *newv, void *state), int flags, void
*initstate, const char *fmtpath, ...);
```

This function behaves precisely like the `maapi_diff_iterate()` function except that it takes an additional format path argument. This path prunes the diff and only changes below the provided path are considered.

```
int maapi_diff_iterate_resume(int sock, enum maapi_iter_ret reply,
enum maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, enum maapi_iter_op
op, confd_value_t *oldv, confd_value_t *newv, void *state), void
*resumestate);
```

The application *must* call this function to finish up the iteration whenever an iterator function for `maapi_diff_iterate()` or `maapi_keypath_diff_iterate()` has returned `ITER_SUSPEND`. If the application does not wish to continue iteration, it must at least call `maapi_diff_iterate_resume(s, ITER_STOP, NULL, NULL)`; to clean up the state. The *reply* parameter is what the iterator function would have returned (i.e. normally `ITER_RECURSE` or `ITER_CONTINUE`) if it hadn't returned `ITER_SUSPEND`. Note that it is up to the iterator function to somehow communicate that it has returned `ITER_SUSPEND` to the caller of `maapi_diff_iterate()` or `maapi_keypath_diff_iterate()`, this can for example be a field in a struct for which a pointer can be passed back and forth via the *state/resumestate* parameters.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADSTATE`.

```
int maapi_iterate(int sock, int thandle, enum maapi_iter_ret (*iter, )
(confd_hkeypath_t *kp, confd_value_t *v, confd_attr_value_t *attr_vals,
int num_attr_vals, void *state), int flags, void *initstate, const char
*fmtpath, ...);
```

This function can be used to iterate over all the data in a transaction and the underlying data store, as opposed to iterating over only the changes like `maapi_diff_iterate()` and `maapi_keypath_diff_iterate()` do. The *fmtpath* parameter can be used to prune the iteration to cover only the subtree below the given path, similar to `maapi_keypath_diff_iterate()` - if *fmtpath* is given as `" / "`, there will not be any such pruning. Additionally, if the flag `MAAPI_FLAG_CONFIG_ONLY` is in effect (see `maapi_set_flags()`), all operational data subtrees will be excluded from the iteration. The *flags* parameter can be given as `ITER_WANT_ATTR` to request attribute values, otherwise it should be 0.

The supplied callback function *iter()* will be called for each node in the data tree included in the iteration. It receives the *kp* parameter which uniquely identifies the node, and if the node is a leaf with a type, also the value of the leaf as the *v* parameter - otherwise *v* is NULL. If the flag `ITER_WANT_ATTR` was given in the call of `maapi_iterate()`, and the node has any attributes set, the *attr_vals* will point to a *num_attr_vals* long array of attributes and values (see `maapi_get_attrs()`), otherwise *attr_vals* is NULL. The return value from *iter()* has the same effect as for `maapi_diff_iterate()`, except that if `ITER_SUSPEND` is returned, the caller then has to call `maapi_iterate_resume()` to continue/finish the iteration.

```
int maapi_iterate_resume(int sock, enum maapi_iter_ret reply, enum
maapi_iter_ret (*iter, )(confd_hkeypath_t *kp, confd_value_t *v,
confd_attr_value_t *attr_vals, int num_attr_vals, void *state), void
*resumestate);
```

The application *must* call this function to finish up the iteration whenever an iterator function for `maapi_iterate()` has returned `ITER_SUSPEND`. If the application does not wish to continue iteration, it must at least call `maapi_iterate_resume(s, ITER_STOP, NULL, NULL)` to clean up the state. The *reply* parameter is what the iterator function would have returned (i.e. normally `ITER_RECURSE` or `ITER_CONTINUE`) if it hadn't returned `ITER_SUSPEND`. Note that it is up to the iterator function to somehow communicate that it has returned `ITER_SUSPEND` to the caller of `maapi_iterate()`, this can for example be a field in a struct for which a pointer can be passed back and forth via the *state/resumestate* parameters.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADSTATE`.

```
int maapi_get_running_db_status(int sock);
```

If a transaction fails in the `commit()` phase, the configuration database is in a possibly inconsistent state. This function queries ConfD on the consistency state. Returns 1 if the configuration is consistent and 0 otherwise.

```
int maapi_set_running_db_status(int sock, int status);
```

This function explicitly sets ConfDs notion of the consistency state.

```
int maapi_list_rollbacks(int sock, struct maapi_rollback *rp, int
*rp_size);
```

List at most `*rp_size` number of rollback files. The number of existing rollback files is reported in `*rp_size` as well. The function will populate an array of `maapi_rollback` structs.

```
int maapi_load_rollback(int sock, int thandle, int rollback_num);
```

Install a rollback file.

```
int maapi_request_action(int sock, confd_tag_value_t *params, int
nparams, confd_tag_value_t **values, int *nvalues, int hashed_ns, const
char *fmt, ...);
```

Invoke an action defined in the data model. The *params* and *values* arrays are the parameters for and results from the action, respectively, and use the Tagged Value Array format described in the [XML STRUCTURES](#) section of the [confd_types\(3\)](#) manual page. The library allocates memory for the result values, and the caller is responsible for freeing it. This can in all cases be done with code like this:

```
confd_tag_value_t *values;
int nvalues = 0, i;

if (maapi_request_action(sock, params, nparams,
                        &values, &nvalues, myprefix_ns,
                        "/path/to/action") == CONFD_OK) {
    ...
    for (i = 0; i < nvalues; i++)
        confd_free_value(CONFD_GET_TAG_VALUE(&values[i]));
    if (nvalues > 0)
        free(values);
}
```

However if the value array is known not to include types that require memory allocation (see `maapi_get_elem()` above), only the array itself needs to be freed.

The socket must have an established user session. The path given by *fmt* and the varargs list is the full path to the action, i.e. the final element must be the name of the action in the data model. Since actions are not associated with ConfD transactions, the namespace must be provided and the path must be absolute - but see `maapi_request_action_th()` below.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_EXTERNAL`

```
int maapi_request_action_th(int sock, int thandle, confd_tag_value_t
*params, int nparams, confd_tag_value_t **values, int *nvalues, const
char *fmt, ...);
```

Does the same thing as `maapi_request_action()`, but uses the current namespace, the path position, and the user session from the transaction indicated by *thandle*, and makes the transaction handle available to the `action()` callback, see [confd_lib_dp\(3\)](#) (this is the only relation to the transaction, and the transaction is not affected in any way by the call itself). This function may be convenient in some cases where actions are invoked in conjunction with a transaction, and it must be used if the action needs to access the transaction store.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_EXTERNAL`

```
int maapi_request_action_str_th(int sock, int thandle, char **output,
const char *cmd_fmt, const char *path_fmt, ...);
```

Does the same thing as `maapi_request_action_th()`, but takes the parameters as a string and returns the result as a string. The library allocates memory for the result string, and the caller is responsible for freeing it. This can in all cases be done with code like this:

```
char *output = NULL;

if (maapi_request_action_str_th(sock, th, &output,
    "test reverse listint [ 1 2 3 4 ]", "/path/to/action") == CONFD_OK) {
    ...
    free(output);
}
```

The varargs in the end of the function must contain all values listed in both format strings (that is *cmd_fmt* and *path_fmt*) in the same order as they occur in the strings. Here follows an equivalent example which uses the format strings:

```
char *output = NULL;

if (maapi_request_action_str_th(sock, th, &output,
    "test %s [ 1 2 3 %d ]", "%s/action",
    "reverse listint", 4, "/path/to") == CONFD_OK) {
    ...
    free(output);
}
```

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_NOSESSION`, `CONFD_ERR_BADPATH`, `CONFD_ERR_NOEXISTS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_EXTERNAL`

```
int maapi_xpath2kpath(int sock, const char *xpath, confd_hkeypath_t
**hkp);
```

Convert a XPath path to a hashed keypath. The XPath expression must be an "instance identifier", i.e. all elements and keys must be fully specified. Namespace prefixes are optional, unless required to resolve ambiguities (e.g. when multiple namespaces have the same root element).

The returned keypath is dynamically allocated, and may further contain dynamically allocated elements. The caller must free the allocated memory, easiest done by calling `confd_free_hkeypath()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH

```
int maapi_user_message(int sock, const char *to, const char *message,  
const char *sender);
```

Send a message to a specific user, a specific user session or all users depending on the *to* parameter. If set to a user name, then *message* will be delivered to all CLI and Web UI sessions by that user. If set to an integer string, eg "10", then *message* will be delivered to that specific user session, CLI or Web UI. If set to "all" then all users will get the *message*.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_sys_message(int sock, const char *to, const char *message);
```

Send a message to a specific user, a specific user session or all users depending on the *to* parameter. If set to a user name, then *message* will be delivered to all CLI and Web UI sessions by that user. If set to an integer string, eg "10", then *message* will be delivered to that specific user session, CLI or Web UI. If set to "all" then all users will get the *message*. No formatting of the message is performed as opposed to the user message where a timestamp and sender information is added to the message.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_prio_message(int sock, const char *to, const char *message);
```

Send a high priority message to a specific user, a specific user session or all users depending on the *to* parameter. If set to a user name, then *message* will be delivered to all CLI and Web UI sessions by that user. If set to an integer string, eg "10", then *message* will be delivered to that specific user session, CLI or Web UI. If set to "all" then all users will get the *message*. No formatting of the message is performed as opposed to the user message where a timestamp and sender information is added to the message.

The message will not be delayed until the user terminates any ongoing command but will be output directly to the terminal without delay. Messages sent using the `maapi_sys_message` and `maapi_user_message`, on the other hand, are not displayed in the middle of some other output but delayed until the any ongoing commands have terminated.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_prompt(int sock, int usess, const char *prompt, int echo,  
char *res, int size);
```

Prompt user for a string. The *echo* parameter is used to control if the input should be echoed or not. If set to CONFD_ECHO all input will be visible and if set to CONFD_NOECHO only stars will be shown instead of the actual characters entered by the user. The resulting string will be stored in *res* and it will be NUL terminated.

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_prompt2(int sock, int usess, const char *prompt, int
echo, int timeout, char *res, int size);
```

This function does the same as `maapi_cli_prompt()`, but also takes a *timeout* parameter, which controls how long (in seconds) to wait for input before aborting.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_prompt_oneof(int sock, int usess, const char *prompt,
char **choice, int count, char *res, int size);
```

Prompt user for one of the strings given in the *choice* parameter. For example:

```
int res;
char buf[BUFSIZ];
char *choice[] = {"yes", "no"};

...

res = maapi_cli_prompt_oneof(sock, uinfo->usid,
                             "Do you want to proceed (yes/no): ",
                             choice, 2, buf, BUFSIZ);
```

The user can enter a unique prefix of the choice but the value returned in *buf* will always be one of the strings provided in the *choice* parameter. The result string stored in *buf* is NUL terminated. If the user enters a value not in *choice* he will automatically be re-prompted. For example:

```
Do you want to proceed (yes/no): maybe
The value must be one of: yes,no.
Do you want to proceed (yes/no):
```

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_prompt_oneof2(int sock, int usess, const char *prompt,
char **choice, int count, int timeout, char *res, int size);
```

This function does the same as `maapi_cli_prompt_oneof()`, but also takes a *timeout* parameter. If no activity is seen for *timeout* seconds an error is returned.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_read_eof(int sock, int usess, int echo, char *res, int
size);
```

Read a multi line string from the CLI. The user has to end the input using ctrl-D. The entered characters will be stored NUL terminated in *res*. The *echo* parameters controls if the entered characters should be echoed or not. If set to CONFD_ECHO they will be visible and if set to CONFD_NOECHO stars will be echoed instead.

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_read_eof2(int sock, int usess, int echo, int timeout,
char *res, int size);
```

This function does the same as `maapi_cli_read_eof()`, but also takes a *timeout* parameter, which indicates how long the user may be idle (in seconds) before the reading is aborted.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_write(int sock, int usess, const char *buf, int size);
```

Write to the CLI.

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_printf(int sock, int usess, const char *fmt, ...);
```

Write to the CLI using printf formatting. This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_vprintf(int sock, int usess, const char *fmt, va_list  
args);
```

Does the same as `maapi_cli_printf()`, but takes a single `va_list` argument instead of a variable number of arguments, like `vprintf()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_accounting(int sock, const char *user, const int usid,  
const char *cmdstr);
```

Generate an audit log entry in the CLI audit log.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_diff_cmd(int sock, int thandle, int thandle_old, char  
*res, int size, int flags, const char *fmt, ...);
```

Get the diff between two sessions as C-/I-style CLI commands.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_path_cmd(int sock, int thandle, char *res, int size, int  
flags, const char *fmt, ...);
```

This function tries to determine which C-/I-style CLI command can be associated with a given path in the data model in context of a given transaction. This is determined by running the formatting code used by the 'show running-config' command for the subtree given by the path, and the looking for text lines associated with the given path. Consequently, if the path does not exist in the transaction no output will be generated, or if `tailf:cli-` annotations have been used to suppress the 'show running-config' text for a path then no such command can be derived.

The *flags* can be given as `MAAPI_FLAG_EMIT_PARENTS` to enable the commands to reach the submode for the path to be emitted.

The *flags* can be given as `MAAPI_FLAG_DELETE` to emit the command to delete the given path.

The *flags* can be given as `MAAPI_FLAG_NON_RECURSIVE` to prevent that all children to a container or list item are displayed.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd_to_path(int sock, const char *line, char *ns, int
nssize, char *path, int psize);
```

Given a data model path formatted as a C- and I-style command, try to determine the corresponding namespace and path. If the string cannot be interpreted as a path an error message is given indicating that the string is either an operational mode command, a configuration mode command, or just badly formatted. The string is interpreted in the context of the current running configuration, ie all xpath expressions in the data model are evaluated in the context of the running config. Note that the same input may result in a correct answer when invoked with one state of the running config, and an error if the running config has another state due to different list elements being present, or xpath (when and display-when) expressions are being evaluated differently.

This function requires that the socket has an established user session.

The *line* is the NUL terminated string of command tokens to be interpreted.

The *ns* and *path* parameters are used for storing the resulting namespace and path.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd_to_path2(int sock, int thandle, const char *line,
char *ns, int nssize, char *path, int psize);
```

Given a data model path formatted as a C- and I-style command, try to determine the corresponding namespace and path. If the string cannot be interpreted as a path an error message is given indicating that the string is either an operational mode command, a configuration mode command, or just badly formatted. The string is interpreted in the context of the provided transaction handler, ie all xpath expressions in the data model are evaluated in the context of the transaction. Note that the same input may result in a correct answer when invoked with one state of one config, and an error when given another config due to different list elements being present, or xpath (when and display-when) expressions are being evaluated differently.

This function requires that the socket has an established user session.

The *th* is a transaction handler.

The *line* is the NUL terminated string of command tokens to be interpreted.

The *ns* and *path* parameters are used for storing the resulting namespace and path.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd(int sock, int usess, const char *buf, int size);
```

Execute CLI command in ongoing CLI session.

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd2(int sock, int usess, const char *buf, int size, int
flags);
```

Execute CLI command in ongoing CLI session.

This function is intended to be called from inside an action callback when invoked from the CLI. The flags field is used to disable certain checks during the execution. The value is a bitmask.

MAAPI_CMD_NO_FULLPATH Do not perform the fullpath check on show commands.

MAAPI_CMD_NO_HIDDEN Allows execution of hidden CLI commands.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd3(int sock, int usess, const char *buf, int size, int flags, const char *unhide, int usize);
```

Execute CLI command in ongoing CLI session.

This function is intended to be called from inside an action callback when invoked from the CLI. The flags field is used to disable certain checks during the execution. The value is a bitmask.

MAAPI_CMD_NO_FULLPATH Do not perform the fullpath check on show commands.

MAAPI_CMD_NO_HIDDEN Allows execution of hidden CLI commands.

The unhide parameter is used for passing a hide group which is unhidden during the execution of the command.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd4(int sock, int usess, const char *buf, int size, int flags, char **unhide, int usize);
```

Execute CLI command in ongoing CLI session.

This function is intended to be called from inside an action callback when invoked from the CLI. The flags field is used to disable certain checks during the execution. The value is a bitmask.

MAAPI_CMD_NO_FULLPATH Do not perform the fullpath check on show commands.

MAAPI_CMD_NO_HIDDEN Allows execution of hidden CLI commands.

The unhide parameter is used for passing hide groups which are unhidden during the execution of the command.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd_io(int sock, int usess, const char *buf, int size, int flags, const char *unhide, int usize);
```

Execute CLI command in ongoing CLI session and output result on socket.

This function is intended to be called from inside an action callback when invoked from the CLI. The flags field is used to disable certain checks during the execution. The value is a bitmask.

MAAPI_CMD_NO_FULLPATH Do not perform the fullpath check on show commands.

MAAPI_CMD_NO_HIDDEN Allows execution of hidden CLI commands.

The unhide parameter is used for passing a hide group which is unhidden during the execution of the command.

The function returns CONFD_ERR on error or a positive integer id that can subsequently be used together with `confd_stream_connect()`. ConfD will write all data in a stream on that socket and when done, ConfD will close its end of the socket.

Once the stream socket is connected we can read the output from the cli command data on the socket. We need to continue reading until we receive EOF on the socket. To check if the command was successful we use the function. `maapi_cli_cmd_io_result()`.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_cli_cmd_io2(int sock, int usess, const char *buf, int size,
int flags, char **unhide, int usize);
```

Execute CLI command in ongoing CLI session and output result on socket.

This function is intended to be called from inside an action callback when invoked from the CLI. The flags field is used to disable certain checks during the execution. The value is a bitmask.

MAAPI_CMD_NO_FULLPATH Do not perform the fullpath check on show commands.

MAAPI_CMD_NO_HIDDEN Allows execution of hidden CLI commands.

The unhide parameter is used for passing hide groups which are unhidden during the execution of the command.

The function returns **CONFD_ERR** on error or a positive integer id that can subsequently be used together with **confd_stream_connect()**. ConfD will write all data in a stream on that socket and when done, ConfD will close its end of the socket.

Once the stream socket is connected we can read the output from the cli command data on the socket. We need to continue reading until we receive EOF on the socket. To check if the command was successful we use the function. **maapi_cli_cmd_io_result()**.

Errors: **CONFD_ERR_MALLOC**, **CONFD_ERR_OS**, **CONFD_ERR_NOEXISTS**

```
int maapi_cli_cmd_io_result(int sock, int id);
```

We use this function to read the status of executing a cli command and streaming the result over a socket. The *sock* parameter must be the same maapi socket we used for **maapi_cli_cmd_io()** and the *id* parameter is the *id* returned by **maapi_cli_cmd_io()**.

Errors: **CONFD_ERR_MALLOC**, **CONFD_ERR_OS**, **CONFD_ERR_ACCESS_DENIED**, **CONFD_ERR_EXTERNAL**

```
int maapi_cli_get(int sock, int usess, const char *opt, char *res, int
size);
```

Read CLI session parameter or attribute.

This function is intended to be called from inside an action callback when invoked from the CLI.

Possible params are complete-on-space, idle-timeout, ignore-leading-space, paginate, "output file", "screen length", "screen width", terminal, history, autowizard, "show defaults", and if enabled, display-level. In addition to this the attributes called annotation, tags and inactive can be read.

Errors: **CONFD_ERR_MALLOC**, **CONFD_ERR_OS**, **CONFD_ERR_NOEXISTS**

```
int maapi_cli_set(int sock, int usess, const char *opt, const char
*value);
```

Set CLI session parameter.

This function is intended to be called from inside an action callback when invoked from the CLI.

Errors: **CONFD_ERR_MALLOC**, **CONFD_ERR_OS**, **CONFD_ERR_NOEXISTS**

```
int maapi_set_readonly_mode(int sock, int flag);
```

There are certain situations where we want to explicitly control if a ConfD instance should be able to handle write operations from the northbound agents. In certain high-availability scenarios we may want to

ensure that a node is a true readonly node, i.e. it should not be possible to initiate new write transactions on that node.

It can also be interesting in upgrade scenarios where we are interested in making sure that no configuration changes can occur during some interval.

This function toggles the readonly mode of a ConfD instance. If the *flag* parameter is non-zero, ConfD will be set in readonly mode, if it is zero, ConfD will be taken out of readonly mode. It is also worth to note that when a ConfD HA node is in slave mode as instructed by the application, no write transactions can occur regardless of the value of the flag set by this function.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_disconnect_remote(int sock, const char *address);
```

Disconnect all remote connections between CONFD_IPC_PORT (see the [ConfD IPC section in the Advanced Topics chapter in the User Guide](#)) and *address*.

Since ConfD clients, e.g. CDB readers/subscribers, are connected using TCP it is also possible to do this remotely over a network. However since TCP doesn't offer a fast and reliable way of detecting that the other end has disappeared ConfD can get stuck waiting for a reply from such a disconnected client.

In some environments there will be an alternative supervision method that can detect when a remote host is unavailable, and in that situation this function can be used to instruct ConfD to drop all remote connections to a particular host. The address parameter is an IP address as a string, and the socket is a maapi socket obtained using `maapi_connect()`. On success, the function returns the number of connections that were closed.



Note

ConfD will close all its sockets with remote address *address*, *except* HA connections. For HA use `confd_ha_slave_dead()` or an HA state transition.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADTYPE, CONFD_ERR_UNAVAILABLE

```
int maapi_disconnect_sockets(int sock, int *sockets, int nsocks);
```

This function is an alternative to `maapi_disconnect_remote()` that can be useful in particular when using the "External IPC" functionality (see "Using a different IPC mechanism" in the [ConfD IPC section in the Advanced Topics chapter in the User Guide](#)). In this case ConfD does not have any knowledge of the remote address of the IPC connections, and thus `maapi_disconnect_remote()` is not applicable. The `maapi_disconnect_sockets()` instead takes an array of *nsocks* socket file descriptor numbers for the *sockets* parameter.

ConfD will close all connected sockets whose local file descriptor number is included the *sockets* array. The file descriptor numbers can be obtained e.g. via the `lssof(8)` command, or some similar tool in case `lssof` does not support the IPC mechanism that is being used.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADTYPE

```
int maapi_save_config(int sock, int thandle, int flags, const char *fmtpath, ...);
```

This function can be used to save the entire config (or a subset thereof) in different formats. The *flags* parameter controls the saving as follows. The value is a bitmask.

MAAPI_CONFIG_XML

The configuration format is XML.

MAAPI_CONFIG_XML_PRETTY

The configuration format is pretty printed XML.

MAAPI_CONFIG_JSON

The configuration is in JSON format.

MAAPI_CONFIG_J

The configuration is in curly bracket Juniper CLI format.

MAAPI_CONFIG_C

The configuration is in Cisco XR style format.

MAAPI_CONFIG_C_IOS

The configuration is in Cisco IOS style format.

MAAPI_CONFIG_XPATH

The *fmlpath* and remaining arguments give an XPath filter instead of a keypath. Can only be used with MAAPI_CONFIG_XML and MAAPI_CONFIG_XML_PRETTY.

MAAPI_CONFIG_WITH_DEFAULTS

Default values are part of the configuration dump.

MAAPI_CONFIG_SHOW_DEFAULTS

Default values are also shown next to the real configuration value. Applies only to the CLI formats.

MAAPI_CONFIG_WITH_OPER

Include operational data in the dump.

MAAPI_CONFIG_HIDE_ALL

Hide all hidden nodes (see below).

MAAPI_CONFIG_UNHIDE_ALL

Unhide all hidden nodes (see below).

MAAPI_CONFIG_WITH_SERVICE_META

Include NCS service-meta-data attributes (refcounter, backpointer, and original-value) in the dump.

MAAPI_CONFIG_NO_PARENTS

When a path is provided its parent nodes are by default included. With this option the output will begin immediately at path - skipping any parents.

MAAPI_CONFIG_OPER_ONLY

Include *only* operational data, and ancestors to operational data nodes, in the dump.

The provided path indicates which part(s) of the configuration to save. By default it is interpreted as a keypath as for other MAAPI functions, and thus identifies the root of a subtree to save. However it is possible to indicate wildcarding of list keys by completely omitting key elements - i.e. this requests save of a subtree for each entry of the corresponding list. For MAAPI_CONFIG_XML and MAAPI_CONFIG_XML_PRETTY it is alternatively possible to give an XPath filter, by including the flag MAAPI_CONFIG_XPATH.

If for example *fmlpath* is `"/aaa:aaa/authentication/users"` we dump a subtree of the AAA data, while if it is `"/aaa:aaa/authentication/users/user/homedir"`, we dump only the `homedir` leaf for each user in the AAA data. If *fmlpath* is NULL, the entire configuration is dumped, except that namespaces with restricted export (from `tailf:export`) are treated as follows:

- When the MAAPI_CONFIG_XML or MAAPI_CONFIG_XML_PRETTY formats are used, the context of the user session that started the transaction is used to select namespaces with restricted export. If the "system" context is used, all namespaces are selected, regardless of export restriction.

- When one of the CLI formats is used, the context used to select namespaces with restricted export is always "cli".

By default, the treatment of nodes with a `tailf:hidden` statement depends on the state of the transaction. For a transaction started via MAAPI, no nodes are hidden, while for a transaction started by another northbound agent (e.g. CLI) and attached to, the nodes that are hidden are the same as in that agent session. The default can be overridden by using one of the flags `MAAPI_CONFIG_HIDE_ALL` and `MAAPI_CONFIG_UNHIDE_ALL`.

The function returns `CONFD_ERR` on error or a positive integer `id` that can subsequently be used together with `confd_stream_connect()`. Thus this function doesn't save the configuration to a file, but rather it returns an integer that is used together with a ConfD stream socket. ConfD will write all data in a stream on that socket and when done, ConfD will close its end of the socket. Thus the following code snippet indicates the usage pattern of this function.

```
int id;
int streamsock;
struct sockaddr_in addr;

id = maapi_save_config(sock, th, flags, path);
if (id < 0) {
    ... handle error ...
}

addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_family = AF_INET;
addr.sin_port = htons(CONFD_PORT);

streamsock = socket(PF_INET, SOCK_STREAM, 0);
confd_stream_connect(streamsock, (struct sockaddr*)&addr,
                    sizeof(struct sockaddr_in), id, 0);
```

Once the stream socket is connected we can read the configuration data on the socket. We need to continue reading until we receive EOF on the socket. To check if the configuration retrieval was successful we use the function `maapi_save_config_result()`.

The stream socket must be connected within 10 seconds after the `id` is received.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BAD_TYPE`

```
int maapi_save_config_result(int sock, int id);
```

We use this function to verify that we received the entire configuration over the stream socket. The `sock` parameter must be the same maapi socket we used for `maapi_save_config()` and the `id` parameter is the `id` returned by `maapi_save_config()`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_EXTERNAL`

```
int maapi_load_config(int sock, int thandle, int flags, const char
*filename);
```

This function loads a configuration from `filename` into ConfD. The `th` parameter is a transaction handle. This can be either for a transaction created by the application, in which case the application must also apply the transaction, or for an attached transaction (which must not be applied by the application). The format of the file can be either XML, curly bracket Juniper CLI format, Cisco XR style format, or Cisco IOS style format. The caller of the function has to indicate which it is by using one of the

MAAPI_CONFIG_XML, MAAPI_CONFIG_J, MAAPI_CONFIG_C, or MAAPI_CONFIG_C_IOS flags, with the same meanings as for `maapi_save_config()`. If the name of the file ends in `.gz` (or `.Z`) then the file is assumed to be gzipped, and will be uncompressed as it is loaded.



Note

If you use a relative pathname for *filename*, it is taken as relative to the working directory of the ConfD daemon, i.e. the directory where the daemon was started.

By default the complete configuration (as allowed by the user of the current transaction) is deleted before the file is loaded. To merge the contents of the file use the `MAAPI_CONFIG_MERGE` flag. To replace only the part of the configuration that is present in the file, use the `MAAPI_CONFIG_REPLACE` flag.

If the transaction *th* is started against the data store `CONFD_OPERATIONAL` config false data is loaded. The existing config false data is not deleted before the file is loaded. Rather it is the responsibility of the client.

The only supported format for loading 'config false' data is `MAAPI_CONFIG_XML`.

Additional flags for `MAAPI_CONFIG_XML`:

`MAAPI_CONFIG_WITH_OPER`

Any operational data in the file should be ignored (instead of producing an error).

`MAAPI_CONFIG_XML_LOAD_LAX`

Lax loading. Ignore unknown namespaces, elements, and attributes.

Additional flag for `MAAPI_CONFIG_C` and `MAAPI_CONFIG_C_IOS`:

`MAAPI_CONFIG_AUTOCOMMIT`

A commit should be performed after each line. In this case the transaction identified by *th* is not used for the loading.

Additional flags for all CLI formats, i.e. `MAAPI_CONFIG_J`, `MAAPI_CONFIG_C`, and `MAAPI_CONFIG_C_IOS`:

`MAAPI_CONFIG_CONTINUE_ON_ERROR`

Do not abort the load when an error is encountered.

`MAAPI_CONFIG_SUPPRESS_ERRORS`

Do not display the long error message but instead a oneline error with the line number.

The other *flags* parameters are the same as for `maapi_save_config()`, however the flags `MAAPI_CONFIG_WITH_SERVICE_META`, `MAAPI_CONFIG_NO_PARENTS`, and `MAAPI_CONFIG_OPER_ONLY` are ignored.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`, `CONFD_ERR_BADPATH`, `CONFD_ERR_BAD_CONFIG`, `CONFD_ERR_ACCESS_DENIED`, `CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_EXTERNAL`

```
int maapi_load_config_cmds(int sock, int thandle, int flags, const char
*cmds, const char *fmt, ...);
```

This function loads a configuration like `maapi_load_config()`, but reads the configuration from the string *cmds* instead of from a file. The *th* and *flags* parameters are the same as for `maapi_load_config()`.

An optional *chroot* path can be given. This is only used with the `MAAPI_CONFIG_C` flag set.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`,
`CONFD_ERR_BADPATH`, `CONFD_ERR_BAD_CONFIG`, `CONFD_ERR_ACCESS_DENIED`,
`CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_EXTERNAL`

```
int maapi_load_config_stream(int sock, int thandle, int flags);
```

This function loads a configuration like `maapi_load_config()`, but reads the configuration from a ConfD stream socket instead of from a file. The *th* and *flags* parameters are the same as for `maapi_load_config()`.

The function returns `CONFD_ERR` on error or a positive integer id that can subsequently be used together with `confd_stream_connect()`. ConfD will read all data from the stream socket until it receives EOF. Thus the following code snippet indicates the usage pattern of this function.

```
int id;
int streamsock;
struct sockaddr_in addr;

id = maapi_load_config_stream(sock, th, flags);
if (id < 0) {
    ... handle error ...
}

addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_family = AF_INET;
addr.sin_port = htons(CONFD_PORT);

streamsock = socket(PF_INET, SOCK_STREAM, 0);
confd_stream_connect(streamsock, (struct sockaddr*)&addr,
    sizeof(struct sockaddr_in), id, 0);
```

Once the stream socket is connected we can write the configuration data on the socket. When we have written the complete configuration, we must close the socket, to make ConfD receive EOF. To check if the configuration load was successful we use the function `maapi_load_config_stream_result()`.

The stream socket must be connected within 10 seconds after the id is received.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`,
`CONFD_ERR_PROTOUSAGE`, `CONFD_ERR_EXTERNAL`

```
int maapi_load_config_stream_result(int sock, int id);
```

We use this function to verify that the configuration we wrote on the stream socket was successfully loaded. The *sock* parameter must be the same maapi socket we used for `maapi_load_config_stream()` and the *id* parameter is the *id* returned by `maapi_load_config_stream()`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADTYPE`,
`CONFD_ERR_BADPATH`, `CONFD_ERR_BAD_CONFIG`, `CONFD_ERR_ACCESS_DENIED`,
`CONFD_ERR_EXTERNAL`

```
int maapi_roll_config(int sock, int thandle, const char *fmtpath, ...);
```

This function can be used to save the equivalent of a rollback file for a given configuration before it is committed (or a subtree thereof) in curly bracket format.

The provided path indicates where we want the configuration to be rooted. It must be a prefix prepended keypath. If *fmtpath* is NULL, a rollback config for the entire configuration is dumped. If for example *fmtpath* is `"/aaa:aaa/authentication/users"` we create a rollback config for a part of the AAA data. It is not possible to extract non-config data using this function.

The function returns `CONFID_ERR` on error or a positive integer *id* that can subsequently be used together with `confd_stream_connect()`. Thus this function doesn't save the rollback configuration to a file, but rather it returns an integer that is used together with a ConfD stream socket. ConfD will write all data in a stream on that socket and when done, ConfD will close its end of the socket. Thus the following code snippet indicates the usage pattern of this function.

```
int id;
int streamsock;
struct sockaddr_in addr;

id = maapi_roll_config(sock, tid, path);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_family = AF_INET;
addr.sin_port = htons(CONFD_PORT);

streamsock = socket(PF_INET, SOCK_STREAM, 0);
confd_stream_connect(streamsock, (struct sockaddr*)&addr,
                    sizeof (struct sockaddr_in), id, 0);
```

Once the stream socket is connected we can read the configuration data on the socket. We need to continue reading until we receive EOF on the socket. To check if the configuration retrieval was successful we use the function `maapi_roll_config_result()`.

The stream socket must be connected within 10 seconds after the *id* is received.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_BAD_TYPE`

```
int maapi_roll_config_result(int sock, int id);
```

We use this function to assert that we received the entire rollback configuration over a stream socket. The *sock* parameter must be the same maapi socket we used for `maapi_roll_config()` and the *id* parameter is the *id* returned by `maapi_roll_config()`.

Errors: `CONFID_ERR_MALLOC`, `CONFID_ERR_OS`, `CONFID_ERR_ACCESS_DENIED`, `CONFID_ERR_EXTERNAL`

```
int maapi_get_stream_progress(int sock, int id);
```

In some cases (e.g. an action or custom command that can be interrupted by the user) it may be useful to be able to terminate ConfD's reading of data from a stream socket (by closing the socket) without waiting for a potentially large amount of data written to the socket to be consumed by ConfD. This function allows us to limit the amount of data "in flight" between the application and ConfD, by reporting the amount of data read by ConfD so far.

The *sock* parameter must be the maapi socket used for a function call that required a stream socket for writing to ConfD (currently the only such function is `maapi_load_config_stream()`), and the *id* parameter is the *id* returned by that function. `maapi_get_stream_progress()` returns the number of bytes that ConfD has read from the stream socket. If *id* does not identify a stream socket that is currently being read by ConfD, the function returns `CONFID_ERR` with `confd_errno` set to `CONFID_ERR_NOEXISTS`. This can be due to e.g. that the socket has been closed, or that an error has occurred - but also that ConfD has determined that all the data has been read (e.g. the end of an XML document has been read). To avoid the latter case, the function should only be called when we have more

data to write, and before the writing of that data. The following code shows a possible way to use this function.

```
#define MAX_IN_FLIGHT 4096

char buf[BUFSIZ];
int sock, streamsock, id;
int n, n_written = 0, n_read = 0;
int result;
...

while (!do_abort() && (n = get_data(buf, sizeof(buf))) > 0) {
    while (n_written - n_read > MAX_IN_FLIGHT) {
        if ((n_read = maapi_get_stream_progress(sock, id)) < 0) {
            ... handle error ...
        }
    }
    if (write(streamsock, buf, n) != n) {
        ... handle error ...
    }
    n_written += n;
}
close(streamsock);
result = maapi_load_config_stream_result(sock, id);
```



Note

A call to `maapi_get_stream_progress()` does not return until the number of bytes read has increased from the previous call (or if there is an error). This means that the above code does not imply busy-looping, but also that if the code was to call `maapi_get_stream_progress()` when `n_read == n_written`, the result would be a deadlock.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_NOEXISTS

```
int maapi_xpath_eval(int sock, int thandle, const char *expr, int
(*result, )(confd_hkeypath_t *kp, confd_value_t *v, void *state), void
(*trace)(char *), void *initstate, const char *fmtpath, ...);
```

This function evaluates the XPath Path expression as supplied in *expr*. For each node in the resulting node set the function *result* is called with the keypath to the resulting node as the first argument, and, if the node is a leaf and has a value, the value of that node as the second argument. The expression will be evaluated using the root node as the context node, unless a path to an existing node is given as the last argument. For each invocation the `result()` function should return `ITER_CONTINUE` to tell the XPath evaluator to continue with the next resulting node. To stop the evaluation the `result()` can return `ITER_STOP` instead.

The *trace* is a pointer to a function that takes a single string as argument. If supplied it will be invoked when the xpath implementation has trace output for the current expression. (For an easy start, for example the `puts(3)` will print the trace output to stdout). If no trace is wanted `NULL` can be given.

The *initstate* parameter can be used for any user supplied opaque data (i.e. whatever is supplied as *initstate* is passed as *state* to the `result()` function for each invocation).

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_XPATH

```
int maapi_xpath_eval_expr(int sock, int thandle, const char *expr, char
**res, void (*trace)(char *), const char *fmtpath, ...);
```

Evaluate the XPath expression given in *expr* and return the result as a string, pointed to by *res*. If the call succeeds, *res* will point to a malloc'ed string that the caller needs to free. If the call fails *res* will be set to NULL.

It is possible to supply a path which will be treated as the initial context node when evaluating *expr* (i.e. if the path is relative, this is treated as the starting point, and this is also the node that *current()* will return when used in the XPath expression). If NULL is given, the current maapi position is used.

The *trace* is a pointer to a function that takes a single string as argument. If supplied it will be invoked when the xpath implementation has trace output for the current expression. (For an easy start, for example the *puts(3)* will print the trace output to stdout). If no trace is wanted NULL can be given.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADPATH, CONFD_ERR_XPATH

```
int maapi_query_start(int sock, int thandle, const char *expr,
const char *context_node, int chunk_size, int initial_offset, enum
confd_query_result_type result_as, int nselect, const char *select[],
int nsort, const char *sort[]);
```

Start a new query attached to the transaction given in *th*. If successful a query handle is returned (the query handle is then used in subsequent calls to *maapi_query_result()* etc). Brief summary of all parameters:

<i>sock</i>	A previously opened maapi socket.
<i>th</i>	A transaction handle to a previously started transaction.
<i>expr</i>	The primary XPath expression.
<i>context_node</i>	The context node (an ikeypath) for the primary expression. NULL is legal, and means that the context node will be /.
<i>chunk_size</i>	How many results to return at a time. If set to 0 a default number will be used.
<i>initial_offset</i>	Which result in line to begin with (1 means to start from the beginning).
<i>result_as</i>	The format the results will be returned in.
<i>nselect</i>	The number of expressions in the <i>select</i> parameter.
<i>select</i>	An array of XPath "select" expressions, of length <i>nselect</i> .
<i>nsort</i>	The number of expressions in the <i>sort</i> parameter.
<i>sort</i>	An array of XPath expressions which will be used for sorting, of length <i>nselect</i> .

A query is a way of evaluating an XPath expression and returning the results in chunks. The usage pattern is as follows: a primary expression is provided in the *expr* argument, which must evaluate to a node-set, the "results". For each node in the results node-set every "select" expression is evaluated with the result node as its context node. For example, given the YANG snippet:

```
list interface {
  key name;
  unique number;
  leaf name {
    type string;
  }
  leaf number {
    type uint32;
    mandatory true;
  }
  leaf enabled {
    type boolean;
```

```

        default true;
    }
    ...
}

```

and given that we want to find the name and number of all enabled interfaces - the *expr* could be `"/interface[enabled='true']"`, and the select expressions would be `{ "name", "number" }`. Note that the select expressions can have any valid XPath expression, so if you wanted to find out an interfaces name, and whether its number is even or not, the expressions would be: `{ "name", "(number mod 2) == 0" }`.

The results are then fetched using the `maapi_query_result()` function, which returns the results on the format specified by the *result_as* parameter. There are four different types of result, as defined by the type enum `confd_query_result_type`:

```

enum confd_query_result_type {
    CONFD_QUERY_STRING = 0,
    CONFD_QUERY_HKEYPATH = 1,
    CONFD_QUERY_HKEYPATH_VALUE = 2,
    CONFD_QUERY_TAG_VALUE = 3
};

```

I.e. the results can be returned as strings, hkeypaths, hkeypaths and values, or tags and values. The string is just the resulting string of evaluating the select XPath expression. For hkeypaths, tags, and values it is the path/tag/value of the *node that the select XPath expression evaluates to*. This means that care must be taken so that the combination of select expression and return types actually yield sensible results (for example `"1 + 2"` is a valid select XPath expression, and would result in the string `"3"` when setting the result type to `CONFD_QUERY_STRING` - but it is not a node, and thus have no hkeypath, tag, or value). A complete example:

```

qh = maapi_query_start(s, th, "/interface[enabled='true']", NULL,
                      1000, 1, CONFD_QUERY_TAG_VALUE,
                      2, (char *[]){ "name", "number" }, 0, NULL);

n = 0;
do {
    maapi_query_result(s, qh, &qr);
    n = qr->nresults;
    for (i=0; i<n; i++) {
        printf("result %d:\n", i + qr->offset);
        for (j=0; j<qr->nelements; j++) {
            // We know the type is tag-value
            char *tag = confd_hash2str(qr->results[i].tv[j].tag.tag);
            confd_pp_value(tmpbuf, BUFSIZ, &qr->results[i].tv[j].v);
            printf("  %s: %s\n", tag, tmpbuf);
        }
    }
    maapi_query_free_result(qr);
} while (n > 0);
maapi_query_stop(s, qh);

```

It is possible to sort the results using the built-in XPath function `sort-by()` (see the [tailf_yang_extensions\(5\)](#) man page)

It is also possible to sort the result using any expressions passed in the *sort* array. These array will be used to construct a temporary index which will live as long as the query is active. For example to start a query sorting first on the enabled leaf, and then on number one would call:

```

qh = maapi_query_start(s, th, "/interface[enabled='true']", NULL,
                      1000, 1, CONFD_QUERY_TAG_VALUE,
                      3, (char *[]){ "name", "number", "enabled" },

```

```

2, (char *[]){ "enabled", "number" });
...

```

Note that the index the query constructs is kept in memory, which will be released when the query is stopped.

```

int maapi_query_result(int sock, int qh, struct confd_query_result
**qrs);

```

Fetch the next available chunk of results associated with query handle *qh*. The results are returned in a struct `confd_query_result`, which is allocated by the library. The structure is defined as:

```

struct confd_query_result {
    enum confd_query_result_type type;
    int offset;
    int nresults;
    int nelements;
    union {
        char **str;
        confd_hkeypath_t *hkp;
        struct {
            confd_hkeypath_t hkp;
            confd_value_t val;
        } *kv;
        confd_tag_value_t *tv;
    } *results;
    void *__internal;          /* confd_lib internal housekeeping */
};

```

The *type* will always be the same as was requested in the call to `maapi_query_start()`, it is there to indicate which of the pointers in the union to use. The *offset* is the number of the first result in this chunk (i.e. for the first chunk it will be 1). How many results that are in this chunk is indicated in *nresults*, when there are no more available results it will be set to 0. Each result consists of *nelements* elements (this number is the same as the number of select parameters given in the call to `maapi_query_start()`).

All data pointed to in the result struct (as well as the struct itself) is allocated by the library - and when finished processing the result the user must call `maapi_query_free_result()` to free this data.

```

int maapi_query_free_result(struct confd_query_result *qrs);

```

The struct `confd_query_result` returned by `maapi_query_result()` is dynamically allocated (and it also contains pointers to other dynamically allocated data) and so it needs to be freed when the result has been processed. Use this function to free the struct `confd_query_result` (and its accompanying data) returned by `maapi_query_result()`.

```

int maapi_query_reset(int sock, int qh);

```

Reset / rewind a running query so that it starts from the beginning again. Next call to `maapi_query_result()` will then return the first chunk of results. The function can be called at any time (i.e. both after all results have been returned to essentially run the same query again, as well as after fetching just one or a couple of results).

```

int maapi_query_reset_to(int sock, int qh, int offset);

```

Like `maapi_query_reset()`, except after the query has been reset it is restarted with the initial offset set to *offset*. Next call to `maapi_query_result()` will then return the first chunk of results at that

offset. The function can be called at any time (i.e. both after all results have been returned to essentially run the same query again, as well as after fetching just one or a couple of results).

```
int maapi_query_stop(int sock, int qh);
```

Stops the running query identified by *qh*, and makes ConfD free up any internal resources associated with the query. If a query isn't explicitly closed using this call it will be cleaned up when the transaction the query is linked to ends.

```
int maapi_install_crypto_keys(int sock);
```

It is possible to define DES3 and AES keys inside `confd.conf`. These keys are used by ConfD to encrypt data which is entered into the system which has either of the two builtin types `tailf:des3-cbc-encrypted-string` or `tailf:aes-cfb-128-encrypted-string`. See [confd_types\(3\)](#).

This function will copy those keys from ConfD (which reads `confd.conf`) into memory in the library. To decrypt data of these types, use the function `confd_decrypt()`, see [confd_lib_lib\(3\)](#).

```
int maapi_do_display(int sock, int thandle, const char *fmtpath, ...);
```

If the data model uses the YANG when or `tailf:display-when` statement, this function can be used to determine if the item given by *fmtpath*, ... should be displayed or not.

```
int maapi_init_upgrade(int sock, int timeoutsecs, int flags);
```

This is the first of three functions that must be called in sequence to perform an in-service data model upgrade, i.e. replace fxs files etc without restarting the ConfD daemon. See the [In-service Data Model Upgrade chapter in the User Guide](#) for a detailed description of this procedure.

This function initializes the upgrade procedure. The *timeoutsecs* parameter specifies a maximum time to wait for users to voluntarily exit from "configure mode" sessions in CLI and Web UI. If transactions are still active when the timeout expires, the function will by default fail with `CONFD_ERR_TIMEOUT`. If the flag `MAAPI_UPGRADE_KILL_ON_TIMEOUT` was given via the *flags* parameter, such transactions will instead be forcibly terminated, allowing the initialization to complete successfully.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_LOCKED`, `CONFD_ERR_BADSTATE`, `CONFD_ERR_HA_WITH_UPGRADE`, `CONFD_ERR_TIMEOUT`, `CONFD_ERR_ABORTED`

```
int maapi_perform_upgrade(int sock, const char **loadpathdirs, int n);
```

When `maapi_init_upgrade()` has completed successfully, this function must be called to instruct ConfD to load the new data model files. The *loadpathdirs* parameter is an array of *n* strings that specify the directories to load from, corresponding to the `/confdConfig/loadPath/dir` elements in `confd.conf` (see [confd.conf\(5\)](#)).

These directories will also be searched for CDB "init files" (see the [CDB chapter in the User Guide](#)). I.e. if the upgrade needs such files, we can place them in one of the new load path directories - or we can include directories that are used *only* for CDB "init files" in the *loadpathdirs* array, corresponding to the `/confdConfig/cdb/initPath/dir` elements that can be specified in `confd.conf`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADSTATE`, `CONFD_ERR_BAD_CONFIG`

```
int maapi_commit_upgrade(int sock);
```

When also `maapi_perform_upgrade()` has completed successfully, this function must be called to make the upgrade permanent. This includes committing the CDB upgrade transaction when CDB is used, and we can thus get all the different validation errors that can otherwise result from `maapi_apply_trans()`.

When `maapi_commit_upgrade()` has completed successfully, the program driving the upgrade must also make sure that the `/confdConfig/loadPath/dir` elements in `confd.conf` reference the new directories. If CDB "init files" are used in the upgrade as described for `maapi_commit_upgrade()` above, the program should also make sure that the `/confdConfig/cdb/initPath/dir` elements reference the directories where those files are located.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_BADSTATE, CONFD_ERR_NOTSET, CONFD_ERR_NON_UNIQUE, CONFD_ERR_BAD_KEYREF, CONFD_ERR_TOO_FEW_ELEMS, CONFD_ERR_TOO_MANY_ELEMS, CONFD_ERR_UNSET_CHOICE, CONFD_ERR_MUST_FAILED, CONFD_ERR_MISSING_INSTANCE, CONFD_ERR_INVALID_INSTANCE, CONFD_ERR_BADTYPE, CONFD_ERR_EXTERNAL

```
int maapi_abort_upgrade(int sock);
```

Calling this function at any point before the call of `maapi_commit_upgrade()` will abort the upgrade.



Note

`maapi_abort_upgrade()` should *not* be called if any of the three previous functions fail - in that case, ConfD will do an internal abort of the upgrade.

CONFD DAEMON CONTROL

```
int maapi_aaa_reload(int sock, int synchronous);
```

When the ConfD AAA tree is populated by an external data provider (see the [AAA chapter in the User Guide](#)), this function can be used by the data provider to notify ConfD when there is a change to the AAA data. I.e. it is an alternative to executing the command **confd --clear-aaa-cache**.

If the *synchronous* parameter is 0, the function will only initiate the loading of the AAA data, just like **confd --clear-aaa-cache** does, and return CONFD_OK as long as the communication with ConfD succeeded. Otherwise it will wait for the loading to complete, and return CONFD_OK only if the loading was successful.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_EXTERNAL

```
int maapi_aaa_reload_path(int sock, int synchronous, const char *fmt, ...);
```

A variant of `maapi_aaa_reload()` that causes only the AAA subtree given by the path in *fmt* to be loaded. This may be useful to load changes to the AAA data when loading the complete AAA tree from an external data provider takes a long time. Obviously care must be taken to make sure that all changes actually get loaded, and a complete load using e.g. `maapi_aaa_reload()` should be done at least when ConfD is started. The path may specify a container or list entry, but not a specific leaf.

Errors: CONFD_ERR_MALLOC, CONFD_ERR_OS, CONFD_ERR_EXTERNAL

```
int maapi_start_phase(int sock, int phase, int synchronous);
```


Once the ConfD daemon has been started in phase0 it is possible to use this function to tell the daemon to proceed to startphase 1 or 2 (as indicated in the *phase* parameter). If *synchronous* is non-zero the call does not return until the daemon has completed the transition to the requested start phase.

Note that start-phase1 can fail, (see documentation of `--start-phase1` in [confd\(1\)](#)) in particular if CDB fails. In that case `maapi_start_phase()` will return `CONFD_ERR`, with `confderrno` set to `CONFD_ERR_START_FAILED`. However if ConfD stops before it has a chance to send back the error `CONFD_EOF` might be returned.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_START_FAILED`

```
int maapi_wait_start(int sock, int phase);
```

To synchronize startup with ConfD this function can be used to wait for ConfD to reach a particular start phase (0, 1, or 2). Note that to implement an equivalent of `confd --wait-started` or `confd --wait-phase0` case must also be taken to retry `maapi_connect()`, which will fail until ConfD has started enough to accept connections to its IPC port.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_PROTOUSAGE`

```
int maapi_stop(int sock, int synchronous);
```

Request the ConfD daemon to stop, if *synchronous* is non-zero the call will wait until ConfD has come to a complete halt. Note that since the daemon exits, the socket won't be re-usable after this call. Equivalent to `confd --stop`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int maapi_reload_config(int sock);
```

Request that the ConfD daemon reloads its configuration files. The daemon will also close and re-open its log files. Equivalent to `confd --reload`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int maapi_reopen_logs(int sock);
```

Request that the ConfD daemon closes and re-opens its log files, useful for logrotate(8).

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`

```
int maapi_rebind_listener(int sock, int listener);
```

Request that the subsystem(s) specified by *listener* rebinds its listener socket(s). Currently open sockets (if any) will be closed, and new sockets created and bound via `bind(2)` and `listen(2)`. This is useful e.g. if `/confdConfig/ignoreBindErrors/enabled` is set to "true" in `confd.conf`, and some bindings have failed due to a problem that subsequently has been fixed. Calling this function then avoids the disable/enable config change that would otherwise be required to cause a rebind.

The following values can be used for the *listener* parameter, ORed together if more than one:

```
#define CONFD_LISTENER_IPC      (1 << 0)
#define CONFD_LISTENER_NETCONF (1 << 1)
#define CONFD_LISTENER_SNMP    (1 << 2)
#define CONFD_LISTENER_CLI     (1 << 3)
#define CONFD_LISTENER_WEBUI   (1 << 4)
```

**Note**

It is not possible to rebind sockets for northbound listeners during the transition from start phase 1 to start phase 2. If this is attempted, the call will fail (and do nothing) with `confd_errno` set to `CONFD_ERR_BADSTATE`.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADSTATE`

```
int maapi_clear_opcache(int sock, const char *fmt, ...);
```

Request clearing of the operational data cache (see the [Operational Data](#) chapter in the User Guide). A path can be given via the *fmt* and subsequent parameters, to clear only the cached data for the subtree designated by that path. To clear the whole cache, pass `NULL` or `"/"` for *fmt*.

Errors: `CONFD_ERR_MALLOC`, `CONFD_ERR_OS`, `CONFD_ERR_BADPATH`

SEE ALSO

`confd_lib(3)` - Confd lib

`confd_types(3)` - ConfD C data types

The ConfD User Guide

Name

confd_types — NSO value representation in C

Synopsis

```
#include <confd_lib.h>
```

DESCRIPTION

The `libconfd` library manages data values such as elements received over the NETCONF protocol. This man page describes how these values as well as the XML paths (`confd_hkeypath_t`) identifying the values are represented in the C language.

TYPEDFS

The following enum defines the different types. These are used to represent data model types from several different sources - see the section [DATA MODEL TYPES](#) at the end of this manual page for a full specification of how the data model types map to these types.

```
enum confd_vtype {
    C_NOEXISTS      = 1, /* end marker */
    C_XMLTAG        = 2, /* struct xml_tag */
    C_SYMBOL        = 3, /* not yet used */
    C_STR           = 4, /* NUL-terminated strings */
    C_BUF           = 5, /* confd_buf_t (string ...) */
    C_INT8          = 6, /* int8_t (int8) */
    C_INT16         = 7, /* int16_t (int16) */
    C_INT32         = 8, /* int32_t (int32) */
    C_INT64         = 9, /* int64_t (int64) */
    C_UINT8         = 10, /* u_int8_t (uint8) */
    C_UINT16        = 11, /* u_int16_t (uint16) */
    C_UINT32        = 12, /* u_int32_t (uint32) */
    C_UINT64        = 13, /* u_int64_t (uint64) */
    C_DOUBLE        = 14, /* double (xs:float,xs:double) */
    C_IPV4          = 15, /* struct in_addr in NBO
                          /* (inet:ipv4-address) */
    C_IPV6          = 16, /* struct in6_addr in NBO
                          /* (inet:ipv6-address) */
    C_BOOL          = 17, /* int (boolean) */
    C_QNAME         = 18, /* struct confd_qname (xs:QName) */
    C_DATETIME      = 19, /* struct confd_datetime
                          /* (yang:date-and-time) */
    C_DATE          = 20, /* struct confd_date (xs:date) */
    C_TIME          = 23, /* struct confd_time (xs:time) */
    C_DURATION      = 27, /* struct confd_duration (xs:duration) */
    C_ENUM_VALUE    = 28, /* int32_t (enumeration) */
    C_BIT32         = 29, /* u_int32_t (bits size 32) */
    C_BIT64         = 30, /* u_int64_t (bits size 64) */
    C_LIST          = 31, /* confd_list (leaf-list) */
    C_XMLBEGIN      = 32, /* struct xml_tag, start of container or
                          /* list entry */
    C_XMLEND        = 33, /* struct xml_tag, end of container or
                          /* list entry */
    C_OBJECTREF     = 34, /* struct confd_hkeypath*
                          /* (instance-identifier) */
    C_UNION         = 35, /* (union) - not used in API functions */
    C_PTR           = 36, /* see cdb_get_values in confd_lib_cdb(3) */
    C_CDBBEGIN      = 37, /* as C_XMLBEGIN, with CDB instance index */
    C_OID           = 38, /* struct confd_snmp_oid* */
}
```

```

/* (yang:object-identifier) */
C_BINARY = 39, /* confd_buf_t (binary ...) */
C_IPV4PREFIX = 40, /* struct confd_ipv4_prefix */
/* (inet:ipv4-prefix) */
C_IPV6PREFIX = 41, /* struct confd_ipv6_prefix */
/* (inet:ipv6-prefix) */
C_DEFAULT = 42, /* default value indicator */
C_DECIMAL64 = 43, /* struct confd_decimal64 (decimal64) */
C_IDENTITYREF = 44, /* struct confd_identityref (identityref) */
C_XMLBEGINDEL = 45, /* as C_XMLBEGIN, but for a deleted list */
/* entry */
C_DQUAD = 46, /* struct confd_dotted_quad */
/* (yang:dotted-quad) */
C_HEXSTR = 47, /* confd_buf_t (yang:hex-string) */
C_IPV4_AND_PLEN = 48, /* struct confd_ipv4_prefix */
/* (tailf:ipv4-address-and-prefix-length) */
C_IPV6_AND_PLEN = 49, /* struct confd_ipv6_prefix */
/* (tailf:ipv6-address-and-prefix-length) */
C_BITBIG = 50, /* confd_buf_t (bits size > 64) */
C_MAXTYPE = /* maximum marker; add new values above */
};

```

A concrete value is represented as a `confd_value_t` C struct:

```

typedef struct confd_value {
    enum confd_vtype type; /* as defined above */
    union {
        struct xml_tag xmltag;
        u_int32_t symbol;
        confd_buf_t buf;
        confd_buf_const_t c_buf;
        char *s;
        const char *c_s;
        int8_t i8;
        int16_t i16;
        int32_t i32;
        int64_t i64;
        u_int8_t u8;
        u_int16_t u16;
        u_int32_t u32;
        u_int64_t u64;
        double d;
        struct in_addr ip;
        struct in6_addr ip6;
        int boolean;
        struct confd_qname qname;
        struct confd_datetime datetime;
        struct confd_date date;
        struct confd_time time;
        struct confd_duration duration;
        int32_t enumvalue;
        u_int32_t b32;
        u_int64_t b64;
        struct confd_list list;
        struct confd_hkeypath *hkp;
        struct confd_vptr ptr;
        struct confd_snmp_oid *oidp;
        struct confd_ipv4_prefix ipv4prefix;
        struct confd_ipv6_prefix ipv6prefix;
        struct confd_decimal64 d64;
        struct confd_identityref idref;
        struct confd_dotted_quad dquad;
        u_int32_t enumhash; /* backwards compat */
    };
};

```

```
    } val;
} confd_value_t;
```

C_NOEXISTS

This is used internally by ConfD, as an end marker in `confd_hkeypath_t` arrays, and as a "value does not exist" indicator in arrays of values.

C_DEFAULT

This is used to indicate that an element with a default value defined in the data model does not have a value set. When reading data from ConfD, we will only get this indication if we specifically request it, otherwise the default value is returned.

C_XMLTAG

An C_XMLTAG value is represented as a struct:

```
struct xml_tag {
    u_int32_t tag;
    u_int32_t ns;
};
```

When a YANG module is compiled by the `confdc(1)` compiler, the `--emit-h` flag is used to generate a `.h` file containing definitions for all the nodes in the module. For example if we compile the following YANG module:

```
# cat blaster.yang
module blaster {
    namespace "http://tail-f.com/ns/blaster";
    prefix blaster;

    import tailf-common {
        prefix tailf;
    }

    typedef Fruit {
        type enumeration {
            enum apple;
            enum orange;
            enum pear;
        }
    }

    container tiny {
        tailf:callpoint xcp;
        leaf foo {
            type int8;
        }
        leaf bad {
            type int16;
        }
    }
}

# confdc -c blaster.yang
# confdc --emit-h blaster.h blaster.fxs
```

We get the following contents in `blaster.h`

```
# cat blaster.h
/*
 * BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE
 * This file has been auto-generated by the confdc compiler.
 * Source: blaster.fxs
 * BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE BEWARE
 */

#ifndef _BLASTER_H_
#define _BLASTER_H_
```

```

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#ifndef blaster__ns
#define blaster__ns 670579579
#define blaster__ns_id "http://tail-f.com/ns/blaster"
#define blaster__ns_uri "http://tail-f.com/ns/blaster"
#endif

#define blaster_orange 1
#define blaster_apple 0
#define blaster_pear 2
#define blaster_foo 161968632
#define blaster_tiny 1046642021
#define blaster_bad 1265139696
#define blaster__callpointid_xcp "xcp"

#ifdef __cplusplus
}
#endif

#endif

```

The integers in the .h file are used in the struct `xml_tag`, thus the container node `tiny` is represented as a `xml_tag` C struct `{tag=1046642021, ns=670579579}` or, using the `#defines` `{tag=blaster_tiny, ns=blaster__ns}`.

Each callpoint, actionpoint, and validate statement also yields a preprocessor symbol. If the symbol is used rather than the literal string in calls to `ConfD`, the C compiler will catch the potential problem when the id in the data model has changed but the C code hasn't been updated.

Sometimes we wish to retrieve a string representation of defined hash values. This can be done with the function `confd_hash2str()`, see the [USING SCHEMA INFORMATION](#) section below.

C_BUF

This type is used to represent the YANG built-in type string and the `xs:token` type. The struct which is used is:

```

typedef struct confd_buf {
    unsigned int size;
    unsigned char *ptr;
} confd_buf_t;

```

Strings passed to the application from `ConfD` are always NUL-terminated. When values of this type are received by the callback functions in [confd_lib_dp\(3\)](#), the `ptr` field is a pointer to `libconfd` private memory, and the data will not survive unless copied by the application.

To create and extract values of type `C_BUF` we do:

```

confd_value_t myval;
char *x; int len;

CONFID_SET_BUF(&myval, "foo", 3)
x = CONFID_GET_BUFPTR(&myval);
len = CONFID_GET_BUFSIZE(&myval);

```

It is important to realize that C_BUF data received by the application through either `maapi_get_elem()` or `cdb_get()` which are of type C_BUF must be freed by the application.

C_STR

This tag is never received by the application. Values and keys received in the various data callbacks (See `confd_register_data_cb()` in [confd_lib_dp\(3\)](#)) never have this type. It is only used when the application replies with values to ConfD. (See `confd_data_reply_value()` in [confd_lib_dp\(3\)](#)).

It is used to represent regular NUL-terminated `char*` values. Example:

```
confd_value_t myval;
myval.type = C_STR;
myval.val.s = "Zaphod";
/* or alternatively and recommended */
CONFID_SET_STR(&myval, "Beeblebrox");
```

C_INT8

Used to represent the YANG built-in type `int8`, which is a signed 8 bit integer. The corresponding C type is `int8_t`. Example:

```
int8_t ival;
confd_value_t myval;

CONFID_SET_INT8(&myval, -32);
ival = CONFID_GET_INT8(&myval);
```

C_INT16

Used to represent the YANG built-in type `int16`, which is a signed 16 bit integer. The corresponding C type is `int16_t`. Example:

```
int16_t ival;
confd_value_t myval;

CONFID_SET_INT16(&myval, -3277);
ival = CONFID_GET_INT16(&myval);
```

C_INT32

Used to represent the YANG built-in type `int32`, which is a signed 32 bit integer. The corresponding C type is `int32_t`. Example:

```
int32_t ival;
confd_value_t myval;

CONFID_SET_INT32(&myval, -77732);
ival = CONFID_GET_INT32(&myval);
```

C_INT64

Used to represent the YANG built-in type `int64`, which is a signed 64 bit integer. The corresponding C type is `int64_t`. Example:

```
int64_t ival;
confd_value_t myval;

CONFID_SET_INT64(&myval, -32);
ival = CONFID_GET_INT64(&myval);
```

C_UINT8

Used to represent the YANG built-in type `uint8`, which is an unsigned 8 bit integer. The corresponding C type is `u_int8_t`. Example:

```
u_int8_t ival;
confd_value_t myval;

CONFID_SET_UINT8(&myval, 32);
ival = CONFID_GET_UINT8(&myval);
```

C_UINT16	<p>Used to represent the YANG built-in type uint16, which is an unsigned 16 bit integer. The corresponding C type is u_int16_t. Example:</p> <pre> u_int16_t ival; confd_value_t myval; CONFDF_SET_UINT16(&myval, 3277); ival = CONFDF_GET_UINT16(&myval); </pre>
C_UINT32	<p>Used to represent the YANG built-in type uint32, which is an unsigned 32 bit integer. The corresponding C type is u_int32_t. Example:</p> <pre> u_int32_t ival; confd_value_t myval; CONFDF_SET_UINT32(&myval, 77732); ival = CONFDF_GET_UINT32(&myval); </pre>
C_UINT64	<p>Used to represent the YANG built-in type uint64, which is an unsigned 64 bit integer. The corresponding C type is u_int64_t. Example:</p> <pre> u_int64_t ival; confd_value_t myval; CONFDF_SET_UINT64(&myval, 32); ival = CONFDF_GET_UINT64(&myval); </pre>
C_DOUBLE	<p>Used to represent the XML schema types xs:decimal, xs:float and xs:double. They are all coerced into the C type double. Example:</p> <pre> double d; confd_value_t myval; CONFDF_SET_DOUBLE(&myval, 3.14); d = CONFDF_GET_DOUBLE(&myval); </pre>
C_BOOL	<p>Used to represent the YANG built-in type boolean. The C representation is an integer with 0 representing false and non-zero representing true. Example:</p> <pre> int bool; confd_value_t myval; CONFDF_SET_BOOL(&myval, 1); b = CONFDF_GET_BOOL(&myval); </pre>
C_QNAME	<p>Used to represent XML Schema type xs:QName which consists of a pair of strings, prefix and a name. Data is allocated by the library as for C_BUF. Example:</p> <pre> unsigned char* prefix, *name; int prefix_len, name_len; confd_value_t myval; CONFDF_SET_QNAME(&myval, "myprefix", 8, "myname", 6); prefix = CONFDF_GET_QNAME_PREFIX_PTR(&myval); prefix_len = CONFDF_GET_QNAME_PREFIX_SIZE(&myval); name = CONFDF_GET_QNAME_NAME_PTR(&myval); name_len = CONFDF_GET_QNAME_NAME_SIZE(&myval); </pre>
C_DATETIME	<p>Used to represent the YANG type yang:date-and-time. The C representation is a struct:</p> <pre> struct confd_datetime { int16_t year; u_int8_t month; </pre>


```

    u_int8_t day;
    u_int8_t hour;
    u_int8_t min;
    u_int8_t sec;
    u_int32_t micro;
    int8_t timezone;
    int8_t timezone_minutes;
};

```

ConfD does not try to convert the data values into timezone independent C structs. The `timezone` and `timezone_minutes` fields are integers where:

<code>timezone == 0 && timezone_minutes == 0</code>	represents UTC. This corresponds to a timezone specification in the string form of "Z" or "+00:00".
<code>-14 <= timezone && timezone <= 14</code>	represents an offset in hours from UTC. In this case <code>timezone_minutes</code> represents a fraction of an hour in minutes if the offset from UTC isn't an integral number of hours, otherwise it is 0. If <code>timezone != 0</code> , its sign gives the direction of the offset, and <code>timezone_minutes</code> is always <code>>= 0</code> - otherwise the sign of <code>timezone_minutes</code> gives the direction of the offset. E.g. <code>timezone == 5 && timezone_minutes == 30</code> corresponds to a timezone specification in the string form of "+05:30".
<code>timezone == CONFD_TIMEZONE_UNDEF</code>	means that the string form indicates lack of timezone information with "-00:00".

It is up to the application to transform these structs into more UNIX friendly structs such as struct `tm` from `<time.h>`. Example:

```

#include <time.h>
confd_value_t myval;
struct confd_datetime dt;
struct tm *tm = localtime(time(NULL));

dt.year = tm->tm_year + 1900; dt.month = tm->tm_mon + 1;
dt.day = tm->tm_mday; dt->hour = tm->tm_hour;
dt.min = tm->tm_min; dt->sec = tm->tm_sec;
dt.micro = 0; dt.timezone = CONFD_TIMEZONE_UNDEF;
CONFD_SET_DATETIME(&myval, dt);
dt = CONFD_GET_DATETIME(&myval);

```

C_DATE

Used to represent the XML Schema type `xs:date`. The C representation is a struct:

```

struct confd_date {
    int16_t year;
    u_int8_t month;
    u_int8_t day;
    int8_t timezone;
    int8_t timezone_minutes;
};

```

Example:

```

confd_value_t myval;
struct confd_date dt;

```

C_TIME	<pre>dt.year = 1960, dt.month = 3, dt.day = 31; dt.timezone = CONFD_TIMEZONE_UNDEF; CONFD_SET_DATE(&myval, dt); dt = CONFD_GET_DATE(&myval);</pre> <p>Used to represent the XML Schema type xs:time. The C representation is a struct:</p>
	<pre>struct confd_time { u_int8_t hour; u_int8_t min; u_int8_t sec; u_int32_t micro; int8_t timezone; int8_t timezone_minutes; };</pre> <p>Example:</p> <pre>confd_value_t myval; struct confd_time dt; dt.hour = 19, dt.min = 3, dt.sec = 31; dt.timezone = CONFD_TIMEZONE_UNDEF; CONFD_SET_TIME(&myval, dt); dt = CONFD_GET_TIME(&myval);</pre>
C_DURATION	<p>Used to represent the XML Schema type xs:duration. The C representation is a struct:</p> <pre>struct confd_duration { u_int32_t years; u_int32_t months; u_int32_t days; u_int32_t hours; u_int32_t mins; u_int32_t secs; u_int32_t micros; };</pre> <p>Example of something that is supposed to last 3 seconds:</p> <pre>confd_value_t myval; struct confd_duration dt; memset(&dt, 0, sizeof(struct confd_duration)); dt.secs = 3; CONFD_SET_DURATION(&myval, dt); dt = CONFD_GET_DURATION(&myval);</pre>
C_IPV4	<p>Used to represent the YANG type inet:ipv4-address. The C representation is a struct in_addr Example:</p> <pre>struct in_addr ip; confd_value_t myval;</pre>
C_IPV6	<p>Used to represent the YANG type inet:ipv6-address. The C representation is a struct in6_addr Example:</p> <pre>struct in6_addr ip6; confd_value_t myval;</pre>

C_ENUM_VALUE

```
inet_pton(AF_INET6, "FFFF::192.168.42.2", &ip6);
CONFD_SET_IPV6(&myval, ip6);
ip6 = CONFD_GET_IPV6(&myval);
```

Used to represent the YANG built-in type enumeration - like the Fruit enumeration from the beginning of this man page.

```
enum fruit {
    ORANGE = blaster_orange,
    APPLE = blaster_apple,
    PEAR = blaster_pear
};

enum fruit f;
confd_value_t myval;
CONFD_SET_ENUM_VALUE(&myval, APPLE);
f = CONFD_GET_ENUM_VALUE(&myval);
```

Thus leafs that have type enumeration in the YANG module do not have values that are strings in the C code, but integer values according to the YANG standard. The file generated by **confdc --emit-h** includes #define symbols for these integer values.

C_BIT32, C_BIT64

Used to represent the YANG built-in type bits when the highest bit position assigned is below 64. In C the value representation for a bitmask is either a 32 bit or a 64 bit unsigned integer, depending on the highest bit position assigned. The file generated by **confdc --emit-h** includes #define symbols giving bitmask values for the defined bit names.

```
u_int32_t mask = 77;
confd_value_t myval;
CONFD_SET_BIT32(&myval, mask);
mask = CONFD_GET_BIT32(&myval);
```

C_BITBIG

Used to represent the YANG built-in type bits when the highest bit position assigned is above 63. In C the value representation for a bitmask in this case is a "little-endian" byte array (confd_buf_t), i.e. byte 0 holds bits 0-7, byte 1 holds bit 8-15, and so on. The file generated by **confdc --emit-h** includes #define symbols giving position values for the defined bit names, as well as the size needed for a byte array that can hold the values for all the defined bits.

```
unsigned char mask[myns__size_mytype];
unsigned char *mask2;
confd_value_t myval;
memset(mask, 0, sizeof(mask));
CONFD_BITBIG_SET_BIT(mask, myns__pos_mytype_somebit);
CONFD_SET_BITBIG(&myval, mask, sizeof(mask));
mask2 = CONFD_GET_BITBIG_PTR(&myval);
```

C_LIST

Used to represent a YANG leaf-list. In C the value representation for is:

```
struct confd_list {
    unsigned int size;
    struct confd_value *ptr;
};
```

Similar to the C_BUF type, the confd library will allocate data when an element of type C_LIST is retrieved via `maapi_get_elem()` or `cdb_get()`. Using `confd_free_value()` (see [confd_lib_lib\(3\)](#)) to free allocated data is especially convenient for C_LIST, as the individual list elements may also have allocated data (e.g. a YANG leaf-list of type string).

To set a value of type `C_LIST` we have to populate the list array separately, for example:

```
confd_value_t arr[5];
confd_value_t v;
confd_value_t *vp;
int i, size;

for (i=0; i<5; i++)
    CONFD_SET_INT32(&arr[i], i);
CONFD_SET_LIST(&v, &arr[0], 5);
```

```
vp = CONFD_GET_LIST(&v);
size = CONFD_GET_LISTSIZE(&v);
```

`C_XMLBEGIN`,
`C_XMLEND`
`C_OBJECTREF`

These are only used in the "Tagged Value Array" format for representing XML structures, see below. The representation is the same as for `C_XMLTAG`.

This is used to represent the YANG built-in type instance-identifier. Values are represented as `confd_hkeypath_t` pointers. Data is allocated by the library as for `C_BUF`. When we read an instance-identifier via e.g. `cdb_get()` we can retrieve the pointer to the keypath as:

```
confd_value_t v;
confd_hkeypath_t *hkp;

cdb_get(sock, &v, mypath);
hkp = CONFD_GET_OBJECTREF(&v);
```

To retrieve the value which is identified by the instance-identifier we can e.g. use the "%h" modifier in the format string used with the CDB and MAAPI API functions.

`C_OID`

This is used to represent the YANG `yang:object-identifier` and `yang:object-identifier-128` types, i.e. SNMP Object Identifiers. The value is a pointer to a struct:

```
struct confd_snmp_oid {
    u_int32_t oid[128];
    int len;
};
```

Data is allocated by the library as for `C_BUF`. When using values of this type, we set or get the `len` element, and the individual OID elements in the `oid` array. This example will store the string "0.1.2" in `buf`:

```
struct confd_snmp_oid myoid;
confd_value_t myval;
char buf[BUFSIZ];
int i;

for (i = 0; i < 3; i++)
    myoid.oid[i] = i;
myoid.len = 3;
CONFD_SET_OID(&myval, &myoid);
```

```
confd_pp_value(buf, sizeof(buf), &myval);
```

`C_BINARY`

This type is used to represent arbitrary binary data. The YANG built-in type `binary`, the ConfD built-in types `tailf:hex-list` and `tailf:octet-list`, and the XML Schema primitive type `xs:hexBinary` all use this type. The value representation

is the same as for C_BUF. Binary (C_BINARY) data received by the application from ConfD is always NUL terminated, but since the data may also contain NUL bytes, it is generally necessary to use the size given by the representation.

```
typedef struct confd_buf {
    unsigned int size;
    unsigned char *ptr;
} confd_buf_t;
```

Data is also allocated by the library as for C_BUF. Example:

```
confd_value_t myval, myval2;
unsigned char *bin;
int len;

bin = CONFD_GET_BINARY_PTR(&myval);
len = CONFD_GET_BINARY_SIZE(&myval);
CONFD_SET_BINARY(&myval2, bin, len);
```

C_IPV4PREFIX

Used to represent the YANG data type inet:ipv4-prefix. The C representation is a struct as follows:

```
struct confd_ipv4_prefix {
    struct in_addr ip;
    u_int8_t len;
};
```

Example:

```
struct confd_ipv4_prefix prefix;
confd_value_t myval;

prefix.ip.s_addr = inet_addr("10.0.0.0");
prefix.len = 8;
CONFD_SET_IPV4PREFIX(&myval, prefix);
prefix = CONFD_GET_IPV4PREFIX(&myval);
```

C_IPV6PREFIX

Used to represent the YANG data type inet:ipv6-prefix. The C representation is a struct as follows:

```
struct confd_ipv6_prefix {
    struct in6_addr ip6;
    u_int8_t len;
};
```

Example:

```
struct confd_ipv6_prefix prefix;
confd_value_t myval;

inet_pton(AF_INET6, "2001:DB8::1428:57A8", &prefix.ip6);
prefix.len = 125;
CONFD_SET_IPV6PREFIX(&myval, prefix);
prefix = CONFD_GET_IPV6PREFIX(&myval);
```

C_DECIMAL64

Used to represent the YANG built-in type decimal64, which is a decimal number with 64 bits of precision. The C representation is a struct as follows:

```
struct confd_decimal64 {
    int64_t value;
    u_int8_t fraction_digits;
};
```

The value element is scaled with the value of the `fraction_digits` element, to be able to represent it as a 64-bit integer. Note that `fraction_digits` is a constant for any given instance of a decimal64 type. It is provided whenever we receive a `C_DECIMAL64` from ConfD. When we provide a `C_DECIMAL64` to ConfD, we can set `fraction_digits` either to the correct value or to 0 - however the value element must always be correctly scaled. See also `confd_get_decimal64_fraction_digits()` in the [confd_lib_lib\(3\)](#) man page.

Example:

```
struct confd_decimal64 d64;
confd_value_t myval;

d64.value = 314159;
d64.fraction_digits = 5;
CONFID_SET_DECIMAL64(&myval, d64);
d64 = CONFID_GET_DECIMAL64(&myval);
```

C_IDENTITYREF

Used to represent the YANG built-in type `identityref`, which references an existing identity. The C representation is a struct as follows:

```
struct confd_identityref {
    u_int32_t ns;
    u_int32_t id;
};
```

The `ns` and `id` elements are hash values that represent the namespace of the module that defines the identity, and the identity within that module.

Example:

```
struct confd_identityref idref;
confd_value_t myval;

idref.ns = des_ns;
idref.id = des_des3;
CONFID_SET_IDENTITYREF(&myval, idref);
idref = CONFID_GET_IDENTITYREF(&myval);
```

C_DQUAD

Used to represent the YANG data type `yang:dotted-quad`. The C representation is a struct as follows:

```
struct confd_dotted_quad {
    unsigned char quad[4];
};
```

Example:

```
struct confd_dotted_quad dquad;
confd_value_t myval;

dquad.quad[0] = 1;
dquad.quad[1] = 2;
dquad.quad[2] = 3;
dquad.quad[3] = 4;
CONFID_SET_DQUAD(&myval, dquad);
dquad = CONFID_GET_DQUAD(&myval);
```

C_HEXSTR

Used to represent the YANG data type `yang:hex-string`. The value representation is the same as for `C_BUF` and `C_BINARY`. `C_HEXSTR` data received by the

application from ConfD is always NUL terminated, but since the data may also contain NUL bytes, it is generally necessary to use the size given by the representation.

```
typedef struct confd_buf {
    unsigned int size;
    unsigned char *ptr;
} confd_buf_t;
```

Data is also allocated by the library as for C_BUF/C_BINARY. Example:

```
confd_value_t myval, myval2;
unsigned char *hex;
int len;

hex = CONFD_GET_HEXSTR_PTR(&myval);
len = CONFD_GET_HEXSTR_SIZE(&myval);
CONFD_SET_HEXSTR(&myval2, bin, len);
```

C_IPV4_AND_PLEN Used to represent the ConfD built-in data type tailf:ipv4-address-and-prefix-length. The C representation is the same struct that is used for C_IPV4PREFIX, as follows:

```
struct confd_ipv4_prefix {
    struct in_addr ip;
    u_int8_t len;
};
```

Example:

```
struct confd_ipv4_prefix ip_and_len;
confd_value_t myval;

ip_and_len.ip.s_addr = inet_addr("172.16.1.2");
ip_and_len.len = 16;
CONFD_SET_IPV4_AND_PLEN(&myval, ip_and_len);
ip_and_len = CONFD_GET_IPV4_AND_PLEN(&myval);
```

C_IPV6_AND_PLEN Used to represent the ConfD built-in data type tailf:ipv6-address-and-prefix-length. The C representation is the same struct that is used for C_IPV6PREFIX, as follows:

```
struct confd_ipv6_prefix {
    struct in6_addr ip6;
    u_int8_t len;
};
```

Example:

```
struct confd_ipv6_prefix ip_and_len;
confd_value_t myval;

inet_pton(AF_INET6, "2001:DB8::1428:57A8", &ip_and_len.ip6);
ip_and_len.len = 64;
CONFD_SET_IPV6_AND_PLEN(&myval, ip_and_len);
ip_and_len = CONFD_GET_IPV6_AND_PLEN(&myval);
```

XML PATHS

Almost all of the callback functions the user is supposed write for the [confd_lib_dp\(3\)](#) library takes a parameter of type `confd_hkeypath_t`. This type includes an array of the type `confd_value_t` described above. The `confd_hkeypath_t` is defined as a C struct:

```
typedef struct confd_hkeypath {
    int len;
    confd_value_t v[MAXDEPTH][MAXKEYLEN];
} confd_hkeypath_t;
```

Where:

```
#define MAXDEPTH 20    /* max depth of data model tree
                        (max KP length + 1) */
#define MAXKEYLEN 9    /* max number of key elems
                        (max keys + 1) */
```

For example, assume we have a YANG module with:

```
container servers {
    tailf:callpoint mycp;
    list server {
        key name;
        max-elements 64;
        leaf name {
            type string;
        }
        leaf ip {
            type inet:ip-address;
        }
        leaf port {
            type inet:port-number;
        }
    }
}
```

Assuming a server entry with the name "www" exists, then the path `/servers/server{www}/ip` is valid and identifies the `ip` leaf in the server entry whose key is "www".

The `confd_hkeypath_t` which corresponds to `/servers/server{www}/ip` is received in reverse order so the following holds assuming the variable holding a pointer to the keypath is called `hkp`.

`hkp->v[0][0]` is the last element, the "ip" element. It is a data model node, and `CONF_GET_XMLTAG(&hkp->v[0][0])` will evaluate to a hashed integer (which can be found in the `confdc` generated .h file as a `#define`)

`hkp->v[1][0]` is the next element in the path. The key element is called "name". This is a string value - thus `strcmp("www", CONF_GET_BUFPTR(&hkp->v[1][0])) == 0` holds.

If we had chosen to use multiple keys in our data model - for example if we had chosen to use both the "name" and the "ip" leafs as keys:

```
key "name ip";
```

The hkeypaths would be different since two keys are required. A valid path identifying a `port` leaf would be `/servers/server{www 10.2.3.4}/port`. In this case we can get to the `ip` part of the key with:

```
struct in_addr ip;
ip = CONF_GET_IPV4(&hkp->v[1][1])
```

USER-DEFINED TYPES

We can define new types in addition to those listed in the TYPEDEFS section above. This can be useful if none of the predefined types, nor a derivation of one of those types via standard YANG restrictions, is

suitable. Of course it is always possible to define a type as a derivation of string and have the application parse the string whenever a value needs to be processed, but with a user-defined type ConfD will do the string <-> value translation just as for the predefined types.

A user-defined type will always have a value representation that uses a `confd_value_t` with one of the enum `confd_vtype` values listed above, but the textual representation and the range(s) of allowed values are defined by the user. The `misc/user_type` example in the collection delivered with the ConfD release shows implementation of several user-defined types - it will be useful to refer to it for the description below.

The choice of `confd_vtype` to use for the value representation can be whatever suits the actual data values best, with one exception:



Note

The `C_LIST` `confd_vtype` value can *not* be used for a leaf that is a key in a YANG list. The "normal" `C_LIST` usage is only for representation of leaf-lists, and a leaf-list can of course not be a key. Thus the ConfD code is not prepared to handle this kind of "value" for a key. It is a strong recommendation to *never* use `C_LIST` for a user-defined type, since even if the type is not initially used for key leafs, subsequent development may see a need for this, at which point it may be cumbersome to change to a different representation.

The example uses `C_INT32`, `C_IPV4PREFIX`, and `C_IPV6PREFIX` for the value representation of the respective types, but in many cases the opaque byte array provided by `C_BINARY` will be most suitable - this can e.g. be mapped to/from an arbitrary C struct.

When we want to implement a user-defined type, we need to specify the type as string, and add a `tailf:typeloint` statement - see [tailf_yang_extensions\(5\)](#). We can use `tailf:typeloint` wherever a built-in or derived type can be specified, i.e. as sub-statement to `typedef`, `leaf`, or `leaf-list`:

```
typedef myType {
    type string;
    tailf:typeloint my_type;
}

container c {
    leaf one {
        type myType;
    }
    leaf two {
        type string;
        tailf:typeloint two_type;
    }
}
```

The argument to the `tailf:typeloint` statement is used to locate the type implementation, similar to how "callpoints" are used to locate data providers, but the actual mechanism is different, as described below.

To actually implement the type definition, we need to write three callback functions that are defined in the struct `confd_type`:

```
struct confd_type {
    /* If a derived type point at the parent */
    struct confd_type *parent;
```

```

/* not used in confspecs, but used in YANG */
struct confd_type *defval;

/* parse value located in str, and validate.
 * returns CONFD_TRUE if value is syntactically correct
 * and CONFD_FALSE otherwise.
 */
int (*str_to_val)(struct confd_type *self,
                  struct confd_type_ctx *ctx,
                  const char *str, unsigned int len,
                  confd_value_t *v);

/* print the value to str.
 * does not print more than len bytes, including trailing NUL.
 * return value as snprintf - i.e. if the value is correct for
 * the type, it returns the length of the string form regardless
 * of the len limit - otherwise it returns a negative number.
 * thus, the NUL terminated output has been completely written
 * if and only if the returned value is nonnegative and less
 * than len.
 * If strp is non-NULL and the string form is constant (i.e.
 * C_ENUM_VALUE), a pointer to the string is stored in *strp.
 */
int (*val_to_str)(struct confd_type *self,
                  struct confd_type_ctx *ctx,
                  const confd_value_t *v,
                  char *str, unsigned int len,
                  const char **strp);

/* returns CONFD_TRUE if value is correct, otherwise CONFD_FALSE
 */
int (*validate)(struct confd_type *self,
                 struct confd_type_ctx *ctx,
                 const confd_value_t *v);

/* data optionally used by the callbacks */
void *opaque;
};

```

I.e. `str_to_val()` and `val_to_str()` are responsible for the string to value and value to string translations, respectively, and `validate()` may be called to verify that a given value adheres to any restrictions on the values allowed for the type. The `errstr` element in the `struct confd_type_ctx *ctx` passed to these functions can be used to return an error message when the function fails - in this case `errstr` must be set to the address of a dynamically allocated string. The other elements in `ctx` are currently unused.

Including user-defined types in a YANG union may need some special consideration. Per the YANG specification, the string form of a value is matched against the union member types in the order they are specified until a match is found, and this procedure determines the type of the value. A corresponding procedure is used by ConfD when the value needs to be converted to a string, but this conversion does not include any evaluation of restrictions etc - the values are assumed to be correct for their type. Thus the `val_to_str()` function for the member types are tried in order until one succeeds, and the resulting string is used. This means that a) `val_to_str()` must verify that the value is of the correct type, i.e. that it has the expected `confd_vtype`, and b) if the value representation is the same for multiple member types, there is no guarantee that the same member type as for the string to value conversion is chosen.

The `opaque` element in the `struct confd_type` can be used for any auxiliary (static) data needed by the functions (on invocation they can reference it as `self->opaque`). The `parent` and `defval` elements are not used in this context, and should be `NULL`.

**Note**

The `str_to_val()` function *must* allocate space (using e.g. `malloc(3)`) for the actual data value for those `confd_value_t` types that are listed as having allocated data above, i.e. `C_BUF`, `C_QNAME`, `C_LIST`, `C_OBJECTREF`, `C_OID`, `C_BINARY`, and `C_HEXSTR`.

We make the implementation available to ConfD by creating one or more shared objects (.so files) containing the above callback functions. Each shared object may implement one or more types, and at startup the ConfD daemon will search the directories specified for `/confdConfig/loadPath` in `confd.conf` for files with a name that match the pattern `"confd_type*.so"` and load them.

Each shared object must also implement an "init" callback:

```
int confd_type_cb_init(struct confd_type_cbs **cbs);
```

When the object has been loaded, ConfD will call this function. It must return a pointer to an array of type callback structures via the `cbs` argument, and the number of elements in the array as return value. The struct `confd_type_cbs` is defined as:

```
struct confd_type_cbs {
    char *typepoint;
    struct confd_type *type;
};
```

These structures are then used by ConfD to locate the implementation of a given type, by searching for a `typepoint` string that matches the `tailf:typepoint` argument in the YANG data model.

**Note**

Since our callbacks are executed directly by the ConfD daemon, it is critically important that they do not have a negative impact on the daemon. No other processing can be done by ConfD while the callbacks are executed, and e.g. a NULL pointer dereference in one of the callbacks will cause ConfD to crash. Thus they should be simple, purely algorithmic functions, never referencing any external resources.

**Note**

When user-defined types are present, the ConfD daemon also needs to load the `libconfd.so` shared library, otherwise used only by applications. This means that either this library must be in one of the system directories that are searched by the OS runtime loader (typically `/lib` and `/usr/lib`), or its location must be given by setting the `LD_LIBRARY_PATH` environment variable before starting ConfD.

The above is enough for ConfD to use the types that we have defined, but the `libconfd` library can also do local string<->value translation if we have loaded the schema information, as described in the [USING SCHEMA INFORMATION](#) section below. For this to work for user-defined types, we must register the type definitions with the library, using one of these functions:

```
int confd_register_ns_type(u_int32_t nshash, const char *name, struct
confd_type *type);
```

Here we must pass the hash value for the namespace where the type is defined as `nshash`, and the name of the type from a `typedef` statement (i.e. *not* the `typepoint` name if they are different) as `name`. Thus we can not use this function to register a user-defined type that is specified "inline" in a `leaf` or `leaf-list` statement, since we don't have a name for the type.

```
int confd_register_node_type(struct confd_cs_node *node, struct
confd_type *type);
```

This function takes a pointer to a schema node (see the section [USING SCHEMA INFORMATION](#)) that uses the type instead of namespace and type name. It is necessary to use this for registration of user-defined types that are specified "inline", but it can also be used for user-defined types specified via typedef. In the latter case it will be equivalent to calling `confd_register_ns_type()` for the typedef, i.e. a single registration will apply to all nodes using the typedef.

The functions can only be called *after* `confd_load_schemas()` or `maapi_load_schemas()` (see below) has been called, and if `confd_load_schemas()/maapi_load_schemas()` is called again, the registration must be re-done. The `misc/user_type` example shows a way to use the exact same code for the shared object and for this registration.

Schema upgrades when the data is stored in CDB requires special consideration for user-defined types. Normally CDB can handle any type changes automatically, and this is true also when changing to/from/ between user-defined types, provided that the following requirements are fulfilled:

- 1 A given typepoint name always refers to the exact same implementation - i.e. same value representation, same range restrictions, etc.
- 2 Shared objects providing implementations for all the typepoint ids used in the new *and* the old schema are made available to ConfD.

I.e. if we change the implementation of a type, we also change the typepoint name, and keep the old implementation around. If requirement 1 isn't fulfilled, we can end up with the case of e.g. a changed value representation between schema versions even though the types are indistinguishable for CDB. This can still be handled by using MAAPI to modify CDB during the upgrade as described in the User Guide, but if that is not done, CDB will just carry the old values over, which in effect results in a corrupt database.

USING SCHEMA INFORMATION

Schema information from the data model can be loaded from the ConfD daemon at runtime using the `maapi_load_schemas()` function, see the [confd_lib_maapi\(3\)](#) manual page. Information for all namespaces loaded into ConfD is then made available. In many cases it may be more convenient to use the `confd_load_schemas()` utility function. For details about this function and those discussed below, see [confd_lib_lib\(3\)](#). After loading the data, we can call `confd_get_nslist()` to find which namespaces are known to the library as a result.

Note that all pointers returned (directly or indirectly) by the functions discussed here reference dynamically allocated memory maintained by the library - they will become invalid if `confd_load_schemas()` or `maapi_load_schemas()` is subsequently called again.

The [confdc\(1\)](#) compiler can also optionally generate a C header file that has #define symbols for the integer values corresponding to data model nodes and enumerations.

When the schema information has been made available to the library, we can format an arbitrary instance of a `confd_value_t` value using `confd_pp_value()` or `confd_ns_pp_value()`, or an arbitrary hkeypath using `confd_pp_kpath()` or `confd_xpath_pp_kpath()`. We can also get a pointer to the string representing a data model node using `confd_hash2str()`.

Furthermore a tree representation of the data model is available, which contains a struct `confd_cs_node` for every node in the data model. There is one tree for each namespace that has toplevel elements.

```
/* flag bits in confd_cs_node_info */
#define CS_NODE_IS_LIST      (1 << 0)
#define CS_NODE_IS_WRITE    (1 << 1)
#define CS_NODE_IS_CDB      (1 << 2)
#define CS_NODE_IS_ACTION   (1 << 3)
#define CS_NODE_IS_PARAM    (1 << 4)
#define CS_NODE_IS_RESULT   (1 << 5)
```

```

#define CS_NODE_IS_NOTIF          (1 << 6)
#define CS_NODE_IS_CASE          (1 << 7)
#define CS_NODE_IS_CONTAINER     (1 << 8)
#define CS_NODE_HAS_WHEN         (1 << 9)
#define CS_NODE_HAS_DISPLAY_WHEN (1 << 10)
#define CS_NODE_HAS_META_DATA    (1 << 11)
#define CS_NODE_IS_WRITE_ALL     (1 << 12)
#define CS_NODE_IS_DYN CS_NODE_IS_LIST /* backwards compat */

/* cmp values in confd_cs_node_info */
#define CS_NODE_CMP_NORMAL        0
#define CS_NODE_CMP_SNMP         1
#define CS_NODE_CMP_SNMP_IMPLIED 2
#define CS_NODE_CMP_USER         3
#define CS_NODE_CMP_UNSORTED     4

struct confd_cs_node_info {
    u_int32_t *keys;
    int minOccurs;
    int maxOccurs; /* -1 if unbounded */
    enum confd_vtype shallow_type;
    struct confd_type *type;
    confd_value_t *defval;
    struct confd_cs_choice *choices;
    int flags;
    u_int8_t cmp;
    struct confd_cs_meta_data *meta_data;
};

struct confd_cs_meta_data {
    char* key;
    char* value;
};

struct confd_cs_node {
    u_int32_t tag;
    u_int32_t ns;
    struct confd_cs_node_info info;
    struct confd_cs_node *parent;
    struct confd_cs_node *children;
    struct confd_cs_node *next;
    void *opaque; /* private user data */
};

struct confd_cs_choice {
    u_int32_t tag;
    u_int32_t ns;
    int minOccurs;
    struct confd_cs_case *default_case;
    struct confd_cs_node *parent; /* NULL if parent is case */
    struct confd_cs_case *cases;
    struct confd_cs_choice *next;
    struct confd_cs_case *case_parent; /* NULL if parent is node */
};

struct confd_cs_case {
    u_int32_t tag;
    u_int32_t ns;
    struct confd_cs_node *first;
    struct confd_cs_node *last;
    struct confd_cs_choice *parent;
    struct confd_cs_case *next;
};

```

```

    struct confd_cs_choice *choices;
};

```

Each `confd_cs_node` is linked to its related nodes: `parent` is a pointer to the parent node, `next` is a pointer to the next sibling node, and `children` is a pointer to the first child node - for each of these, a NULL pointer has the obvious meaning.

Each `confd_cs_node` also contains an information structure: For a list node, the `keys` field is a zero-terminated array of integers - these are the `tag` values for the children nodes that are key elements. This makes it possible to find the name of a key element in a keypath. If the `confd_cs_node` is not a list node, the `keys` field is NULL. The `shallow_type` field gives the "primitive" type for the element, i.e. the enum `confd_vtype` value that is used in the `confd_value_t` representation.

Typed leaf nodes also carry a complete type definition via the `type` pointer, which can be used with the `conf_str2val()` and `confd_val2str()` functions, as well as the leaf's default value (if any) via the `defval` pointer.

If the YANG `choice` statement is used in the data model, additional structures are created by the schema loading. For list and container nodes that have `choice` statements, the `choices` element in `confd_cs_node_info` is a pointer to a linked list of `confd_cs_choice` structures representing the choices. Each `confd_cs_choice` has a pointer to the parent node and a `cases` pointer to a linked list of `confd_cs_case` structures representing the cases for that choice. Finally, each `confd_cs_case` structure has pointers to the parent `confd_cs_choice` structure, and to the `confd_cs_node` structures representing the first and last element in the case. Those `confd_cs_node` structures, i.e. the "toplevel" elements of a case, have the `CS_NODE_IS_CASE` flag set. Note that it is possible for a case to be "empty", i.e. there are no elements in the case - then the `first` and `last` pointers in the `confd_cs_case` structure are NULL.

For a list node, the sort order is indicated by the `cmp` element in `confd_cs_node_info`. The value `CS_NODE_CMP_NORMAL` means an ordinary, system ordered, list. `CS_NODE_CMP_SNMP` is system ordered, but ordered according to SNMP lexicographical order, and `CS_NODE_CMP_SNMP IMPLIED` is an SNMP lexicographical order where the last key has an IMPLIED keyword. `CS_NODE_CMP_UNSORTED` is system ordered, but is not sorted. The value `CS_NODE_CMP_USER` denotes an "ordered-by user" list.

If the `tailf:meta-data` extension is used for a node, the `meta_data` element points to an array of struct `confd_cs_meta_data`, otherwise it is NULL. In the array, the `key` element is the argument of `tailf:meta-data`, and the `value` element is the argument of the `tailf:meta-value` substatement, if any - otherwise it is NULL. The end of the array is indicated by a struct where the `key` element is NULL.

Action and notification specifications are included in the tree in the same way as the `config/data` elements - they are indicated by the `CS_NODE_IS_ACTION` flag being set on the `action` node, and the `CS_NODE_IS_NOTIF` flag being set on the `notification` node, respectively. Furthermore the nodes corresponding to the sub-statements of the action's `input` statement have the `CS_NODE_IS_PARAM` flag set, and those corresponding to the sub-statements of the action's `output` statement have the `CS_NODE_IS_RESULT` flag set. Note that the `input` and `output` statements do not have corresponding nodes in the tree.

The `confd_find_cs_root()` function returns the root of the tree for a given namespace, and the `confd_find_cs_node()`, `confd_find_cs_node_child()`, and `confd_cs_node_cd()` functions are useful for navigating the tree. Assume that we have the following data model:

```

container servers {
  list server {
    key name;
    max-elements 64;
    leaf name {

```

```

        type string;
    }
    leaf ip {
        type inet:ip-address;
    }
    leaf port {
        type inet:port-number;
    }
}
}

```

Then, given the keypath `/servers/server{www}` in `confd_hkeypath_t` form, a call to `confd_find_cs_node()` would return a struct `confd_cs_node`, i.e. a pointer into the tree, as in:

```

struct confd_cs_node *csp;
char *name;
csp = confd_find_cs_node(mykeypath, mykeypath->len);
name = confd_hash2str(csp->info.keys[0])

```

and the C variable `name` will have the value `"name"`. These functions make it possible to format keypaths in various ways.

If we have a keypath which identifies a node below the one we are interested in, such as `/servers/server{www}/ip`, we can use the `len` parameter as in `confd_find_cs_node(kp, 3)` where 3 is the length of the keypath we wish to consider.

The equivalent of the above `confd_find_cs_node()` example, but using a string keypath, could be written as:

```

csp = confd_cs_node_cd(confd_find_cs_root(mynamespace),
    "/servers/server{www}");

```

The `type` field in the struct `confd_cs_node_info` can be used for data model aware string <-> value translations. E.g. assuming that we have a `confd_hkeypath_t *kp` representing the element `/servers/server{www}/ip`, we can do the following:

```

confd_value_t v;
csp = confd_find_cs_node(kp, kp->len);
confd_str2val(csp->info.type, "10.0.0.1", &v);

```

The `confd_value_t v` will then be filled in with the corresponding `C_IPV4` value. This technique is generally necessary for translating `C_ENUM_VALUE` values to the corresponding strings (or vice versa), since there isn't a type-independent mapping. But `confd_val2str()` (or `confd_str2val()`) can always do the translation, since it is given the full type information. E.g. this will store the string `"nonVolatile"` in `buf`:

```

confd_value_t v;
char buf[64];

CONF_SET_ENUM_VALUE(&v, 3);
root = confd_find_cs_root(SNMP_COMMUNITY_MIB__ns);
csp = confd_cs_node_cd(root, "/SNMP-COMMUNITY-MIB/snmpCommunityTable/"
    "snmpCommunityEntry/snmpCommunityStorageType");
confd_val2str(csp->info.type, &v, buf, sizeof(buf));

```

The type information can also be found by using the `confd_find_ns_type()` function to look up the type name as a string in the namespace where it is defined - i.e. we could alternatively have achieved the same result with:

```

CONF_SET_ENUM_VALUE(&v, 3);
type = confd_find_ns_type(SNMPv2_TC__ns, "StorageType");
confd_val2str(type, &v, buf, sizeof(buf));

```

If we give 0 for the *nshash* argument to `confd_find_ns_type()`, the type name will be looked up among the ConfD built-in types (i.e. the YANG built-in types, the types defined in the YANG "tailf-common" module, and the types defined in the pre-defined "confd" and/or "xs" namespaces) - e.g. the type information for `/servers/server{www}/name` could be found with `confd_find_ns_type(0, "string")`.

XML STRUCTURES

Two different methods are used to represent a subtree of data nodes. "Value Array" describes a format that is simpler but has some limitations, while "Tagged Value Array" describes a format that is more complex but can represent an arbitrary subtree.

Value Array

The simpler format is an array of `confd_value_t` elements corresponding to the complete contents of a list entry or container. The content of sub-list entries cannot be represented. The array is populated through a "depth first" traversal of the data tree as follows:

- 1 Optional leafs or presence containers that do not exist use a single array element, with type `C_NOEXISTS` (value ignored).
- 2 List nodes use a single array element, with type `C_NOEXISTS` (value ignored), regardless of the actual number of entries or their contents.
- 3 Leafs with a type other than empty use an array element with their type and value as usual.
- 4 Leafs of type empty use an array element with type `C_XMLTAG`, and `tag` and `ns` set according to the leaf name.
- 5 Containers use one array element with type `C_XMLTAG`, and `tag` and `ns` set according to the element name, followed by array elements for the sub-nodes according to this list.

Note that the list or container node corresponding to the complete array is not included in the array, and that there is no array element for the "end" of a container.

As an example, the array corresponding to the `/servers/server{www}` list entry above could be populated as:

```
confd_value_t v[3];
struct in_addr ip;

CONFSET_STR(&v[0], "www");
ip.s_addr = inet_addr("192.168.1.2");
CONFSET_IPV4(&v[1], ip);
CONFSET_UINT16(&v[2], 80);
```

Tagged Value Array

This format uses an array of `confd_tag_value_t` elements. This is a structure defined as:

```
typedef struct confd_tag_value {
    struct xml_tag tag;
    confd_value_t v;
} confd_tag_value_t;
```

I.e. each value element is associated with the struct `xml_tag` that identifies the node in the data model. The `ns` element of the struct `xml_tag` can normally be set to 0, with the meaning "current namespace". The array is populated, normally through a "depth first" traversal of the data tree, as follows:

- 1 Optional leafs or presence containers that do not exist are omitted entirely from the array.

- 2 List and container nodes use one array element where the value has type `C_XMLBEGIN`, and `tag` and `ns` set according to the node name, followed by array elements for the sub-nodes according to this list, followed by one array element where the value has type `C_XMLEND`, and `tag` and `ns` set according to the node name.
- 3 Leafs with a type other than empty use an array element with their type and value as usual.
- 4 Leafs of type empty use an array element where the value has type `C_XMLTAG`, and `tag` and `ns` set according to the leaf name.

Note that the list or container node corresponding to the complete array is not included in the array. In some usages, non-optional nodes may also be omitted from the array - refer to the relevant API documentation to see whether this is allowed and the semantics of doing so.

A set of `CONF_SET_TAG_XXX()` macros corresponding to the `CONF_SET_XXX()` macros described above are provided - these set the `ns` element to 0 and the `tag` element to their second argument. The array corresponding to the `/servers/server{www}` list entry above could be populated as:

```
confd_tag_value_t tv[3];
struct in_addr ip;

CONF_SET_TAG_STR(&tv[0], servers_name, "www");
ip.s_addr = inet_addr("192.168.1.2");
CONF_SET_TAG_IPV4(&tv[1], servers_ip, ip);
CONF_SET_TAG_UINT16(&tv[2], servers_port, 80);
```

There are also macros to access the components of the `confd_tag_value_t` elements:

```
confd_tag_value_t tv;
u_int16_t port;

if (CONF_GET_TAG_TAG(&tv) == servers_port)
    port = CONF_GET_UINT16(CONF_GET_TAG_VALUE(&tv));
```

DATA MODEL TYPES

This section describes the types that can be used in YANG data modeling, and their C representation. Also listed is the corresponding SMIV2 type, which is used when a data model is translated into a MIB. In several cases, the data model type cannot easily be translated into a native SMIV2 type. In those cases, the type OCTET STRING is used in the translation. The SNMP agent in ConfD will in those cases send the string representation of the value over SNMP. For example, the `xs:float` value `3.14` is sent as the string `"3.14"`.

These subsections describe the following sets of types, which can be used with YANG data modeling:

- [YANG built-in types](#)
- [The ietf-yang-types YANG module](#)
- [The ietf-inet-types YANG module](#)
- [The tailf-common YANG module](#)
- [The tailf-xsd-types YANG module](#)

YANG built-in types

These types are built-in to the YANG language, and also built-in to ConfD.

<code>int8</code>	A signed 8-bit integer. <ul style="list-style-type: none">• <code>value.type = C_INT8</code>• union element = <code>i8</code>
-------------------	--

	<ul style="list-style-type: none"> • C type = <code>int8_t</code> • SMIV2 type = Integer32 (-128 .. 127)
<code>int16</code>	<p>A signed 16-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_INT16</code> • union element = <code>i16</code> • C type = <code>int16_t</code> • SMIV2 type = Integer32 (-32768 .. 32767)
<code>int32</code>	<p>A signed 32-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_INT32</code> • union element = <code>i32</code> • C type = <code>int32_t</code> • SMIV2 type = Integer32
<code>int64</code>	<p>A signed 64-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_INT64</code> • union element = <code>i64</code> • C type = <code>int64_t</code> • SMIV2 type = OCTET STRING
<code>uint8</code>	<p>An unsigned 8-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT8</code> • union element = <code>u8</code> • C type = <code>u_int8_t</code> • SMIV2 type = Unsigned32 (0 .. 255)
<code>uint16</code>	<p>An unsigned 16-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT16</code> • union element = <code>u16</code> • C type = <code>u_int16_t</code> • SMIV2 type = Unsigned32 (0 .. 65535)
<code>uint32</code>	<p>An unsigned 32-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT32</code> • union element = <code>u32</code> • C type = <code>u_int32_t</code> • SMIV2 type = Unsigned32
<code>uint64</code>	<p>An unsigned 64-bit integer.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT64</code> • union element = <code>u64</code> • C type = <code>u_int64_t</code> • SMIV2 type = OCTET STRING
<code>decimal64</code>	<p>A decimal number with 64 bits of precision. The C representation uses a struct with a 64-bit signed integer for the scaled value, and an unsigned 8-bit integer in the range 1..18 for the number of fraction digits specified by the <code>fraction-digits</code> sub-statement.</p> <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_DECIMAL64</code> • union element = <code>d64</code> • C type = <code>struct confd_decimal64</code> • SMIV2 type = OCTET STRING

string	<p>The string type is represented as a struct <code>confd_buf_t</code> when <i>received</i> from ConfD in the C code. I.e. it is NUL-terminated and also has a size given. However, when the C code wants to produce a value of the string type it is possible to use a <code>confd_value_t</code> with the value type <code>C_BUF</code> or <code>C_STR</code> (which requires a NUL-terminated string)</p> <ul style="list-style-type: none"> • <code>value.type = C_BUF</code> • union element = <code>buf</code> • C type = <code>confd_buf_t</code> • SMIV2 type = OCTET STRING
boolean	<p>The boolean values "true" and "false".</p> <ul style="list-style-type: none"> • <code>value.type = C_BOOL</code> • union element = <code>boolean</code> • C type = <code>int</code> • SMIV2 type = <code>TruthValue</code>
enumeration	<p>Enumerated strings with associated numeric values. The C representation uses the numeric values.</p> <ul style="list-style-type: none"> • <code>value.type = C_ENUM_VALUE</code> • union element = <code>enumvalue</code> • C type = <code>int32_t</code> • SMIV2 type = <code>INTEGER</code>
bits	<p>A set of bits or flags. Depending on the highest argument given to a <code>position</code> sub-statement, the C representation uses either <code>C_BIT32</code>, <code>C_BIT64</code>, or <code>C_BITBIG</code>.</p> <ul style="list-style-type: none"> • <code>value.type = C_BIT32, C_BIT64, or C_BITBIG</code> • union element = <code>b32, b64, or buf</code> • C type = <code>u_int32_t, u_int64_t, or confd_buf_t</code> • SMIV2 type = <code>Unsigned32 or OCTET STRING</code>
binary	<p>Any binary data.</p> <ul style="list-style-type: none"> • <code>value.type = C_BINARY</code> • union element = <code>buf</code> • C type = <code>confd_buf_t</code> • SMIV2 type = <code>OCTET STRING</code>
identityref	<p>A reference to an abstract identity.</p> <ul style="list-style-type: none"> • <code>value.type = C_IDENTITYREF</code> • union element = <code>idref</code> • C type = <code>struct confd_identityref</code> • SMIV2 type = <code>OCTET STRING</code>
union	<p>The union type has no special <code>confd_value_t</code> representation - elements are represented as one of the member types according to the current value instantiation. This means that for unions that comprise different "primitive" types, applications must check the <code>type</code> element to determine the type, and the type safe alternatives to the <code>cdb_get()</code> and <code>maapi_get_elem()</code> functions can not be used.</p> <p>Note that the YANG specification stipulates that when a value of type union is validated, the <i>first</i> matching member type should be chosen. Consider this YANG fragment:</p>

```
leaf uni {
    type union {
        type int32;
        type int64;
    }
}
```

If we set the leaf to the value 2, it should thus be of type int32, not type int64. This is enforced when ConfD converts a string to an internal value, but not when setting values "directly" via e.g. `maapi_set_elem()` or `cdb_set_elem()`. It is thus possible to set the leaf to a `C_INT64` with the value 2, but this is formally an invalid value.

Applications setting values of type union must thus take care to choose the member type correctly, or alternatively provide the value as a string via one of the functions `maapi_set_elem2()`, `cdb_set_elem2()`, or `confd_str2val()`. These functions will always turn the string "2" into a `C_INT32` with the above definition.

The SMIV2 type is an OCTET STRING.

instance-identifier

The instance-identifier built-in type is used to uniquely identify a particular instance node in the data tree. The syntax for an instance-identifier is a subset of the XPath abbreviated syntax.

- `value.type = C_OBJECTREF`
- union element = `hkp`
- C type = `confd_hkeypath_t`
- SMIV2 type = OCTET STRING

The leaf-list statement


The values of a YANG `leaf-list` node is represented as an element with a list of values of the type given by the `type` sub-statement.

- `value.type = C_LIST`
- union element = `list`
- C type = `struct confd_list`
- SMIV2 type = OCTET STRING

The ietf-yang-types YANG module

This module contains a collection of generally useful derived YANG data types. They are defined in the `urn:ietf:params:xml:ns:yang:ietf-yang-types` namespace.

<code>yang:counter32</code> , <code>yang:zero-based-counter32</code>	32-bit counters, corresponding to the Counter32 type and the ZeroBasedCounter32 textual convention of the SMIV2. <ul style="list-style-type: none"> • <code>value.type = C_UINT32</code> • union element = <code>u32</code> • C type = <code>u_int32_t</code> • SMIV2 type = Counter32
<code>yang:counter64</code> , <code>yang:zero-based-counter64</code>	64-bit counters, corresponding to the Counter64 type and the ZeroBasedCounter64 textual convention of the SMIV2. <ul style="list-style-type: none"> • <code>value.type = C_UINT64</code> • union element = <code>u64</code>

	<ul style="list-style-type: none"> • C type = <code>u_int64_t</code> • SMIV2 type = Counter64
<code>yang:gauge32</code>	32-bit gauge value, corresponding to the Gauge32 type of the SMIV2. <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT32</code> • union element = <code>u32</code> • C type = <code>u_int32_t</code> • SMIV2 type = Counter32
<code>yang:gauge64</code>	64-bit gauge value, corresponding to the CounterBasedGauge64 SMIV2 textual convention. <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT64</code> • union element = <code>u64</code> • C type = <code>u_int64_t</code> • SMIV2 type = Counter64
<code>yang:object-identifier</code> , <code>yang:object-identifier-128</code>	An SNMP OBJECT IDENTIFIER (OID). This is a sequence of integers which identifies an object instance for example "1.3.6.1.4.1.24961.1".
<div style="display: flex; align-items: center;">  <div> <p>Note</p> <p>The <code>tailf:value-length</code> restriction is measured in integer elements for <code>object-identifier</code> and <code>object-identifier-128</code>.</p> </div> </div>	
	<ul style="list-style-type: none"> • <code>value.type</code> = <code>C_OID</code> • union element = <code>oidp</code> • C type = <code>confd_snmp_oid</code> • SMIV2 type = OBJECT IDENTIFIER
<code>yang:yang-identifier</code>	A YANG identifier string as defined by the 'identifier' rule in Section 12 of RFC 6020. <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_BUF</code> • union element = <code>buf</code> • C type = <code>confd_buf_t</code> • SMIV2 type = OCTET STRING
<code>yang:date-and-time</code>	The date-and-time type is a profile of the ISO 8601 standard for representation of dates and times using the Gregorian calendar. <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_DATETIME</code> • union element = <code>datetime</code> • C type = <code>struct confd_datetime</code> • SMIV2 type = DateAndTime
<code>yang:timeticks</code> , <code>yang:timestamp</code>	Time ticks and time stamps, measured in hundredths of seconds. Corresponding to the TimeTicks type and the TimeStamp textual convention of the SMIV2. <ul style="list-style-type: none"> • <code>value.type</code> = <code>C_UINT32</code> • union element = <code>u32</code> • C type = <code>u_int32_t</code> • SMIV2 type = Counter32

yang:phys-address

Represents media- or physical-level addresses represented as a sequence octets, each octet represented by two hexadecimal digits. Octets are separated by colons.



Note

The `tailf:value-length` restriction is measured in number of octets for phys-address.

- `value.type = C_BINARY`
- union element = `buf`
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

yang:mac-address

The mac-address type represents an IEEE 802 MAC address. The length of the ConfD C_BINARY representation is always 6.

- `value.type = C_BINARY`
- union element = `buf`
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

yang:xpath1.0

This type represents an XPATH 1.0 expression.

- `value.type = C_BUF`
- union element = `buf`
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

yang:hex-string

A hexadecimal string with octets represented as hex digits separated by colons.



Note

The `tailf:value-length` restriction is measured in number of octets for hex-string.

- `value.type = C_HEXSTR`
- union element = `buf`
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

yang:uuid

A Universally Unique Identifier in the string representation defined in RFC 4122.

- `value.type = C_BUF`
- union element = `buf`
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

yang:dotted-quad

An unsigned 32-bit number expressed in the dotted-quad notation.

- `value.type = C_DQUAD`
- union element = `dquad`
- C type = `struct confd_dotted_quad`
- SMIV2 type = OCTET STRING

The ietf-inet-types YANG module

This module contains a collection of generally useful derived YANG data types for Internet addresses and related things. They are defined in the `urn:ietf:params:xml:ns:yang:ietf-inet-types` namespace.

<code>inet:ip-version</code>	<p>This value represents the version of the IP protocol.</p> <ul style="list-style-type: none">• <code>value.type = C_ENUM_VALUE</code>• <code>union element = enumvalue</code>• C type = <code>int32_t</code>• SMIV2 type = <code>INTEGER</code>
<code>inet:dscp</code>	<p>The <code>dscp</code> type represents a Differentiated Services Code-Point.</p> <ul style="list-style-type: none">• <code>value.type = C_UINT8</code>• <code>union element = u8</code>• C type = <code>u_int8_t</code>• SMIV2 type = <code>Unsigned32 (0 .. 255)</code>
<code>inet:ipv6-flow-label</code>	<p>The <code>flow-label</code> type represents flow identifier or Flow Label in an IPv6 packet header.</p> <ul style="list-style-type: none">• <code>value.type = C_UINT32</code>• <code>union element = u32</code>• C type = <code>u_int32_t</code>• SMIV2 type = <code>Unsigned32</code>
<code>inet:port-number</code>	<p>The <code>port-number</code> type represents a 16-bit port number of an Internet transport layer protocol such as UDP, TCP, DCCP or SCTP.</p>
<code>inet:as-number</code>	<p>The value space and representation is identical to the built-in <code>uint16</code> type.</p> <p>The <code>as-number</code> type represents autonomous system numbers which identify an Autonomous System (AS).</p>
<code>inet:ip-address</code>	<p>The value space and representation is identical to the built-in <code>uint32</code> type.</p> <p>The <code>ip-address</code> type represents an IP address and is IP version neutral. The format of the textual representations implies the IP version.</p> <p>This is a union of the <code>inet:ipv4-address</code> and <code>inet:ipv6-address</code> types defined below. The representation is thus identical to the representation for one of these types.</p>
<code>inet:ipv4-address</code>	<p>The SMIV2 type is an OCTET STRING (SIZE (4 16)).</p> <p>The <code>ipv4-address</code> type represents an IPv4 address in dotted-quad notation. The use of a zone index is not supported by ConfD.</p> <ul style="list-style-type: none">• <code>value.type = C_IPV4</code>• <code>union element = ip</code>• C type = <code>struct in_addr</code>• SMIV2 type = <code>IpAddress</code>
<code>inet:ipv6-address</code>	<p>The <code>ipv6-address</code> type represents an IPv6 address in full, mixed, shortened and shortened mixed notation.</p> <p>The use of a zone index is not supported by ConfD.</p> <ul style="list-style-type: none">• <code>value.type = C_IPV6</code>• <code>union element = ip6</code>

	<ul style="list-style-type: none"> • C type = struct in6_addr • SMIV2 type = IPV6-MIB:Ipv6Address
inet:ip-prefix	<p>The ip-prefix type represents an IP prefix and is IP version neutral. The format of the textual representations implies the IP version.</p> <p>This is a union of the inet:ipv4-prefix and inet:ipv6-prefix types defined below. The representation is thus identical to the representation for one of these types.</p>
inet:ipv4-prefix	<p>The SMIV2 type is an OCTET STRING (SIZE (5 17)).</p> <p>The ipv4-prefix type represents an IPv4 address prefix. The prefix length is given by the number following the slash character and must be less than or equal to 32.</p> <p>A prefix length value of n corresponds to an IP address mask which has n contiguous 1-bits from the most significant bit (MSB) and all other bits set to 0.</p> <p>The IPv4 address represented in dotted quad notation must have all bits that do not belong to the prefix set to zero.</p> <p>An example: 10.0.0.0/8</p>
inet:ipv6-prefix	<ul style="list-style-type: none"> • value.type = C_IPV4PREFIX • union element = ipv4prefix • C type = struct confd_ipv4_prefix • SMIV2 type = OCTET STRING (SIZE (5)) <p>The ipv6-prefix type represents an IPv6 address prefix. The prefix length is given by the number following the slash character and must be less than or equal 128.</p> <p>A prefix length value of n corresponds to an IP address mask which has n contiguous 1-bits from the most significant bit (MSB) and all other bits set to 0.</p> <p>The IPv6 address must have all bits that do not belong to the prefix set to zero.</p> <p>An example: 2001:DB8::1428:57AB/125</p>
inet:domain-name	<ul style="list-style-type: none"> • value.type = C_IPV6PREFIX • union element = ipv6prefix • C type = struct confd_ipv6_prefix • SMIV2 type = OCTET STRING (SIZE (17)) <p>The domain-name type represents a DNS domain name. The name SHOULD be fully qualified whenever possible.</p>
inet:host	<ul style="list-style-type: none"> • value.type = C_BUF • union element = buf • C type = confd_buf_t • SMIV2 type = OCTET STRING <p>The host type represents either an IP address or a DNS domain name.</p> <p>This is a union of the inet:ip-address and inet:domain-name types defined above. The representation is thus identical to the representation for one of these types.</p>

inet:uri

The SMIV2 type is an OCTET STRING, which contains the textual representation of the domain name or address.

The uri type represents a Uniform Resource Identifier (URI) as defined by STD 66.

- `value.type = C_BUF`
- union element = buf
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

The iana-crypt-hash YANG module

This module defines a type for storing passwords using a hash function, and features to indicate which hash functions are supported by an implementation. The type is defined in the `urn:ietf:params:xml:ns:yang:iana-crypt-hash` namespace.

ianach:crypt-hash

The crypt-hash type is used to store passwords using a hash function.

The algorithms for applying the hash function and encoding the result are implemented in various UNIX systems as the function `crypt(3)`. A value of this type matches one of the forms:

```
$0$<clear text password>
$<id>$<salt>$<password hash>
$<id>$<parameter>$<salt>$<password hash>
```

The "\$0\$" prefix indicates that the value is clear text. When such a value is received by the server, a hash value is calculated, and the string "\$<id>\$<salt>\$" or "\$<id>\$<parameter>\$<salt>\$" is prepended to the result. This value is stored in the configuration data store.

If a value starting with "\$<id>\$", where <id> is not "0", is received, the server knows that the value already represents a hashed value, and stores it as is in the data store.

In the Tail-f implementation, this type is logically a union of the types `tailf:md5-digest-string`, `tailf:sha-256-digest-string`, and `tailf:sha-512-digest-string` - see the section [The tailf-common YANG module](#) below. All the hashed values of these types are accepted, and the choice of algorithm to use for hashing clear text is specified via the `/confdConfig/cryptHash/algorithm` parameter in `confd.conf` (see [confd.conf\(5\)](#)). If the algorithm is set to "sha-256" or "sha-512", it can be tuned via the `/confdConfig/cryptHash/rounds` parameter in `confd.conf`.

- `value.type = C_BUF`
- union element = buf
- C type = `confd_buf_t`
- SMIV2 type = OCTET STRING

The tailf-common YANG module

This module defines Tail-f common YANG types, that are built-in to ConfD.

tailf:size

A value that represents a number of bytes. An example could be `S1G8M7K956B`; meaning 1GB+8MB+7KB+956B = 1082138556 bytes. The value must start with an S. Any byte magnifier can be left

tailf:octet-list



Note

out, i.e. S1K1B equals 1025 bytes. The order is significant though, i.e. S1B56G is not a valid byte size.

The value space and representation is identical to the built-in uint64 type.

A list of dot-separated octets for example "192.168.255.1.0".

The `tailf:value-length` restriction is measured in number of octets for octet-list.

tailf:hex-list



Note

- `value.type = C_BINARY`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

A list of colon-separated hexa-decimal octets for example "4F:4C:41:71".

The `tailf:value-length` restriction is measured in octets of binary data for hex-list.

tailf:md5-digest-string

- `value.type = C_BINARY`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

The md5-digest-string type automatically computes a MD5 digest for a value adhering to this type.

This is best explained using an example. Suppose we have a leaf:

```
leaf key {  
    type tailf:md5-digest-string;  
}
```

A valid configuration is:

```
<key>$0$In god we trust.</key>
```

The "\$0\$" prefix indicates that this is plain text and that this value should be represented as a MD5 digest from now. ConfD computes a MD5 digest for the value and prepends "\$1\$<salt>\$", where <salt> is a random eight character salt used to generate the digest. When this value later on is fetched from ConfD the following is returned:

```
<key>$1$fB$ndk2z/PIS0S1SvzWLqTJb.</key>
```

A value adhering to md5-digest-string must have "\$0\$" or a "\$1\$<salt>\$" prefix.

The digest algorithm is the same as the md5 crypt function used for encrypting passwords for various UNIX systems, e.g. <http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/lib/libcrypt/crypt.c?rev=1.5&content-type=text/plain>

**Note**

The pattern restriction can not be used with this type.

tailf:sha-256-digest-string

- `value.type = C_BUF`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

The sha-256-digest-string type automatically computes a SHA-256 digest for a value adhering to this type. A value of this type matches one of the forms:

```
$0$<clear text password>
$5$<salt>$<password hash>
$5$rounds=<number>$<salt>$<password hash>
```

The "\$0\$" prefix indicates that this is plain text. When a plain text value is received by the server, a SHA-256 digest is calculated, and the string "\$5\$<salt>\$" is prepended to the result, where <salt> is a random 16 character salt used to generate the digest. This value is stored in the configuration data store. The algorithm can be tuned via the `/confdConfig/cryptHash/rounds` parameter in `confd.conf` (see [confd.conf\(5\)](#)), which if set to a number other than the default will cause "\$5\$rounds=<number>\$<salt>\$" to be prepended instead of only "\$5\$<salt>\$".

If a value starting with "\$5\$" is received, the server knows that the value already represents a SHA-256 digest, and stores it as is in the data store.

The digest algorithm used is the same as the SHA-256 crypt function used for encrypting passwords for various UNIX systems, see e.g. <http://www.akkadia.org/drepper/SHA-crypt.txt>

- `value.type = C_BUF`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

tailf:sha-512-digest-string

The sha-512-digest-string type automatically computes a SHA-512 digest for a value adhering to this type. A value of this type matches one of the forms:

```
$0$<clear text password>
$6$<salt>$<password hash>
$6$rounds=<number>$<salt>$<password hash>
```

The "\$0\$" prefix indicates that this is plain text. When a plain text value is received by the server, a SHA-512 digest is calculated, and the string "\$6\$<salt>\$" is prepended to the result, where <salt> is a random 16 character salt used to generate the digest. This value is stored in the configuration data store. The algorithm can be tuned via the `/confdConfig/cryptHash/rounds` parameter in `confd.conf` (see [confd.conf\(5\)](#)), which if set to a number other

than the default will cause "\$6\$rounds=<number>\$<salt>\$" to be prepended instead of only "\$6\$<salt>\$".

If a value starting with "\$6\$" is received, the server knows that the value already represents a SHA-512 digest, and stores it as is in the data store.

The digest algorithm used is the same as the SHA-512 crypt function used for encrypting passwords for various UNIX systems, see e.g. <http://www.akkadia.org/drepper/SHA-crypt.txt>

- `value.type = C_BUF`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

`tailf:des3-cbc-encrypted-string`

The `des3-cbc-encrypted-string` type automatically encrypts a value adhering to this type using DES in CBC mode followed by a base64 conversion. If the value isn't encrypted already, that is.

This is best explained using an example. Suppose we have a leaf:

```
leaf enc {  
    type tailf:des3-cbc-encrypted-string;  
}
```

A valid configuration is:

```
<enc>$0$In god we trust.</enc>
```

The "\$0\$" prefix indicates that this is plain text. When a plain text value is received by the server, the value is DES3/Base64 encrypted, and the string "\$7\$" is prepended. The resulting string is stored in the configuration data store.

When a value of this type is read, the encrypted value is always returned. In the example above, the following value could be returned:

```
<enc>$7$Qxxsn8BVzxphCdflqRwZm6noKKmt0QoSWnRnhcXqocg=</enc>
```

If a value starting with "\$7\$" is received, the server knows that the value is already encrypted, and stores it as is in the data store.

A value adhering to this type must have a "\$0\$" or a "\$7\$" prefix.

ConfD uses a configurable set of encryption keys to encrypt the string. For details, see the description of the `encryptedStrings` configurable in the [confd.conf\(5\)](#) manual page.



Note

The `pattern` restriction can not be used with this type.

- `value.type = C_BUF`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

tailf:aes-cfb-128-encrypted-string

The aes-cfb-128-encrypted-string works exactly like des3-cbc-encrypted-string but AES/128bits in CFB mode is used to encrypt the string. The prefix for encrypted values is "\$8\$".



Note

The pattern restriction can not be used with this type.

tailf:ip-address-and-prefix-length

- `value.type = C_BUF`
- `union element = buf`
- `C type = confd_buf_t`
- `SMIv2 type = OCTET STRING`

The ip-address-and-prefix-length type represents a combination of an IP address and a prefix length and is IP version neutral. The format of the textual representations implies the IP version.

This is a union of the tailf:ipv4-address-and-prefix-length and tailf:ipv6-address-and-prefix-length types defined below. The representation is thus identical to the representation for one of these types.

tailf:ipv4-address-and-prefix-length

The SMIv2 type is an OCTET STRING (SIZE (5|17)).

The ipv4-address-and-prefix-length type represents a combination of an IPv4 address and a prefix length. The prefix length is given by the number following the slash character and must be less than or equal to 32.

An example: 172.16.1.2/16

- `value.type = C_IPV4_AND_PLEN`
- `union element = ipv4prefix`
- `C type = struct confd_ipv4_prefix`
- `SMIv2 type = OCTET STRING (SIZE (5))`

tailf:ipv6-address-and-prefix-length

The ipv6-address-and-prefix-length type represents a combination of an IPv6 address and a prefix length. The prefix length is given by the number following the slash character and must be less than or equal to 128.

An example: 2001:DB8::1428:57AB/64

- `value.type = C_IPV6_AND_PLEN`
- `union element = ipv6prefix`
- `C type = struct confd_ipv6_prefix`
- `SMIv2 type = OCTET STRING (SIZE (17))`

The tailf-xsd-types YANG module

"This module contains useful XML Schema Datatypes that are not covered by YANG types directly.

xs:duration

- `value.type = C_DURATION`
- `union element = duration`
- `C type = struct confd_duration`
- `SMIv2 type = OCTET STRING`

xs:date

- `value.type = C_DATE`

xs:time	<ul style="list-style-type: none"> • union element = date • C type = struct confd_date • SMIV2 type = OCTET STRING • value.type = C_TIME • union element = time • C type = struct confd_time • SMIV2 type = OCTET STRING
xs:token	<ul style="list-style-type: none"> • value.type = C_BUF • union element = buf • C type = confd_buf_t • SMIV2 type = OCTET STRING
xs:hexBinary	<ul style="list-style-type: none"> • value.type = C_BINARY • union element = buf • C type = confd_buf_t • SMIV2 type = OCTET STRING
xs:QName	<ul style="list-style-type: none"> • value.type = C_QNAME • union element = qname • C type = struct confd_qname • SMIV2 type = <not applicable>
xs:decimal, xs:float, xs:double	<ul style="list-style-type: none"> • value.type = C_DOUBLE • union element = d • C type = double • SMIV2 type = OCTET STRING

SEE ALSO

The NSO User Guide

confd_lib(3) - confd C library.

confd.conf(5) - confd daemon configuration file format

NCS man-pages, Volume 5



Name

clispec — CLI specification file format

DESCRIPTION

This manual page describes the syntax and semantics of a NSO CLI specification file (from now on called "clispec"). A clispec is an XML configuration file describing commands to be added to the automatically rendered Juniper and Cisco style NSO CLI. It also makes it possible to modify the behavior of standard/built-in commands, using move/delete operations and customizable confirmation prompts. In Cisco style custom mode-specific commands can be added by specifying a mount point relating to the specified mode.



Tip

In the NSO distribution there is an Emacs mode suitable for clispec editing.

A clispec file (with a .cli suffix) is to be compiled using the **ncsc** compiler into an internal representation (with a .ccl suffix), ready to be loaded by the NSO daemon on startup. Like this:

```
$ ncsc -c commands.cli
$ ls commands.ccl
commands.ccl
```

The .ccl file should be put in the NSO daemon loadPath as described in [ncs.conf\(5\)](#). When the NSO daemon is started the clispec is loaded accordingly.

The NSO daemon loads all .ccl files it finds on startup. I.e, you can have one or more clispec files for Cisco XR (C) style CLI emulation, one or more for Cisco IOS (I), and one or more for Juniper (J) style emulation. If you drop several .ccl files in the loadPath all will be loaded. The standard commands are defined in ncs.cli (available in the NSO distribution). The intention is that we use ncs.cli as a starting point, i.e. first we delete, reorder and replace built-in commands (if needed) and we then proceed to add our own custom commands.

EXAMPLE

The ncs-light.cli example is a light version of the standard ncs.cli. It adds one operational mode command and one configure mode command, implemented by two OS executables, it also removes the 'save' command from the pipe commands.

Example 10. ncs-light.cli

```
<clispec xmlns="http://tail-f.com/ns/clispec/1.0" style="j">
  <operationalMode>
    <modifications>
      <delete src="file"/>
      <confirmText src="quit">
        Are you really sure you want to quit?
      </confirmText>
      <help src="configure private">Edit a private copy of the configuration</help>
      <info src="configure private">Edit a private copy of the configuration</info>
    </modifications>

    <cmd name="copy" mount="file">
      <info>Copy a file</info>
      <help>Copy a file in the file system.</help>
      <callback>
        <exec>
          <osCommand>cp</osCommand>
        </exec>
      </callback>
    </cmd>
  </operationalMode>
</clispec>
```

```

        <options>
          <uid>confd</uid>
        </options>
      </exec>
    </callback>
  <params>
    <param>
      <type><file/></type>
      <info>&lt;source file&gt;</info>
    </param>
    <param>
      <type><file/></type>
      <info>&lt;destination&gt;</info>
    </param>
  </params>
</cmd>
</operationalMode>

<configureMode>
  <cmd name="adduser" mount="wizard">
    <info>Create a user</info>
    <help>Create a user and assign him/her to a group.</help>
    <callback>
      <exec>
        <osCommand>adduser.sh</osCommand>
      </exec>
    </callback>
  </cmd>
</configureMode>

<pipeCmds>
  <modifications>
    <delete src="save"/>
  </modifications>
</pipeCmds>
</clispec>

```

ncs-light.cli achieves the following:

- Adds a confirmation prompt to the standard operation "delete" command.
- Deletes the standard "file" command.
- Adds the operational mode command "copy" and mounts it under the standard "file" command.
- The "copy" command is implemented using the OS executable "/usr/bin/cp".
- The executable is called with parameters as defined by the "params" element.
- The executable runs as the same user id as NSO as defined by the "uid" element.
- Adds the configure command "adduser" and mounts it under the standard "wizard" command.

Below we present the gory details when it comes to constructs in a clispec.

ELEMENTS AND ATTRIBUTES

This section lists all clispec elements and their attributes including their type (within parentheses) and default values (within square brackets). Elements are written using a path notation to make it easier to see how they relate to each other.

Note: \$MODE is either "operationalMode", "configureMode" or "pipeCmds".

```
/clispec
```

This is the top level element which contains (in order) zero or more "operationalMode" elements, zero or more "configureMode" element, and zero or more "pipeCmds" elements.

`/clispec/$MODE`

The \$MODE ("operationalMode", "configureMode", or "pipeCmds") element contains (in order) zero or one "modifications" elements, zero or more "start" elements, zero or more "show" elements, and zero or more "cmd" elements.

The "show" elements are only used in the C-style CLI.

It has a name attribute which is used to create a named custom mode. A custom command can be defined for entering custom modes. See the cmd/callback/mode elements below.

`/clispec/$MODE/modifications`

The "modifications" element describes which operations to apply to the built-in commands. It contains (in any order) zero or more "delete", "move", "paginate", "info", "paraminfo", "help", "paramhelp", "confirmText", "defaultConfirmOption", "dropElem", "compactElem", "compactStatsElem", "columnStats", "multiValue", "columnWidth", "columnAlign", "defaultColumnAlign", "noKeyCompletion", "noMatchCompletion", "modeName", "suppressMode", "suppressTable", "enforceTable", "showTemplate", "showTemplateLegend", "showTemplateEnter", "showTemplateFooter", "runTemplate", "runTemplateLegend", "runTemplateEnter", "runTemplateFooter", "addMode", "autocommitDelay", "keymap", "pipeFlags", "addPipeFlags", "negPipeFlags", "legend", "footer", "suppressKeyAbbrev", "allowKeyAbbrev", "hasRange", "suppressRange", "allowWildcard", "suppressWildcard", "suppressValidationWarningPrompt", "displayEmptyConfig", "displayWhen", "customRange", "completion", "keepKeyOrder" and "simpleType" elements.

`/clispec/$MODE/modifications/paginate`

The "paginate" element can be used to change the default paginate behavior for a built-in command.

Attributes:

<i>path</i> (cmdpathType)	The "path" attribute is mandatory. It specifies which command to change. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
<i>value</i> (true false)	The "value" attribute is mandatory. It specifies whether the paginate attribute should be enabled or disabled by default.

`/clispec/$MODE/modifications/displayWhen`

The "displayWhen" element can be used to add a displayWhen xpath condition to a command.

Attributes:

<i>path</i> (cmdpathType)	The "path" attribute is mandatory. It specifies which command to change. cmdpathType is a space-
---------------------------	--

<i>expr</i> (xpath expression)	separated list of commands, pointing out a specific sub-command. The "expr" attribute is mandatory. It specifies an xpath expression. If the expression evaluates to true then the command is available, otherwise not.
<i>ctx</i> (path)	The "ctx" attribute is optional. If not specified the current editpath/mode-path is used as context node for the xpath evaluation. Note that the xpath expression will automatically evaluate to false if a display when expression is used for a top-level command and no ctx is specified. The path may contain variables defined in the dict.

/clispec/\$MODE/modifications/move

The "move" element can be used to move (rename) a built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to move. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
<i>dest</i> (cmdpathType)	The "dest" attribute is mandatory. It specifies where to move the command specified by the "src" attribute. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
<i>inclSubCmds</i> (xs:boolean)	The "inclSubCmds" attribute is optional. If specified and set to true then all commands to which the 'src' command is a prefix command will be included in the move operation.

An example:

```
<configureMode>
  <modifications>
    <move src="load" dest="xload" inclSubCmds="true">
  </modifications>
</configureMode>
```

would in the C-style CLI move 'load', 'load merge', 'load override' and 'load replace' to 'xload', 'xload merge', 'xload override' and 'xload replace', respectively.

/clispec/\$MODE/modifications/copy

The "copy" element can be used to copy a built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to copy. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	---

dest (cmdpathType)

The "dest" attribute is mandatory. It specifies where to copy the command specified by the "src" attribute. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

inclSubCmds (xs:boolean)

The "inclSubCmds" attribute is optional. If specified and set to true then all commands to which the 'src' command is a prefix command will be included in the copy operation.

An example:

```
<configureMode>
  <modifications>
    <copy src="load" dest="xload" inclSubCmds="
  </modifications>
</configureMode>
```

would in the C-style CLI copy 'load', 'load merge', 'load override' and 'load replace' to 'xload', 'xload merge', 'xload override' and 'xload replace', respectively.

/clispec/\$MODE/modifications/delete

The "delete" element makes it possible to delete a built-in command. Note that commands that are auto-rendered from the data model cannot be removed using this modification. To remove an auto-rendered command use the 'tailf:hidden' element in the data model.

Attributes:

src (cmdpathType)

The "src" attribute is mandatory. It specifies which command to delete. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

/clispec/\$MODE/modifications/pipeFlags

The "pipeFlags" element makes it possible to modify the pipe flags of the builtin commands. The argument is a space separated list of pipe flags. It will replace the builtin list.

Attributes:

src (cmdpathType)

The "src" attribute is mandatory. It specifies which command to modify. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

/clispec/\$MODE/modifications/addPipeFlags

The "addPipeFlags" element makes it possible to add pipe flags to the existing list of pipe flags for a builtin command. The argument is a space separated list of pipe flags.

Attributes:

src (cmdpathType)

The "src" attribute is mandatory. It specifies which command to modify. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

/clispec/\$MODE/modifications/negPipeFlags

The "negPipeFlags" element makes it possible to modify the neg pipe flags of the builtin commands. The argument is a space separated list of neg pipe flags. It will replace the builtin list.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to modify. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	---

/clispec/\$MODE/modifications/columnWidth

The "columnWidth" element can be used to set fixed widths for specific columns in auto-rendered tables.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to set the column width for. pathType is a space-separated list of node names, pointing out a specific data model node.
<i>width</i> (xs:positiveInteger)	The "width" attribute is mandatory. It specified a fixed column width.

Note that the tailf:cli-column-width YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/columnAlign

The "columnAlign" element can be used to specify the alignment of the data in specific columns in auto-rendered tables.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to set the column alignment for. pathType is a space-separated list of node names, pointing out a specific data model node.
<i>align</i> (left right center)	The "align" attribute is mandatory.

Note that the tailf:cli-column-align YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/defaultColumnAlign

The "defaultColumnAlign" element can be used to specify a default alignment of a simpletype when used in auto-rendered tables.

Attributes:

<i>namespace</i> (xs:string)	The "namespace" attribute is required. It specifies in which namespace the type is found. It can be either the namespace URI or the namespace prefix.
------------------------------	---

<i>name</i> (xs:string)	The "name" attribute is required. It specifies the name of the type in the given namespace.
<i>align</i> (left right center)	The "align" attribute is mandatory.

`/clispec/$MODE/modifications/multiLinePrompt`

The "multiLinePrompt" element can be used to specify that the CLI should automatically enter multi-line prompt mode when prompting for values of the given type.

Attributes:

<i>namespace</i> (xs:string)	The "namespace" attribute is required. It specifies in which namespace the type is found. It can be either the namespace URI or the namespace prefix.
<i>name</i> (xs:string)	The "name" attribute is required. It specifies the name of the type in the given namespace.

`/clispec/$MODE/modifications/runTemplate`

The "run" element is used for specifying a template to use by the "show running-config" command in the C- and I-style CLIs. The syntax is the same as for the showTemplate above. The template is only used if it is associated with a leaf element. Containers and lists cannot have runTemplates.

Note that extreme care must be taken when using this feature if the result should be paste:able into the CLI again.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to apply the show running-config template. pathType is a space-separated list of elements, pointing out a specific container element.
------------------------	---

Note that the tailf:cli-run-template YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/runTemplateLegend`

The "runTemplateLegend" element is used for specifying a template to use by the show running-config command in the C- and I-style CLIs when displaying a set of list nodes as a legend.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to apply the show running-config template. pathType is a space-separated list of elements, pointing out a specific container element.
------------------------	---

Note that the tailf:cli-run-template-legend YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/runTemplateEnter`

The "runTemplateEnter" element is used for specifying a template to use by the show running-config command in the C- and I-style CLIs when displaying a set of list element nodes before displaying each instance.

In addition to the builtin variables in ordinary templates there are two additional variables available: *.prefix_str* and *.key_str*.

<i>.prefix_str</i>	The <i>.prefix_str</i> variable contains the text displayed before the key values when auto-rendering an enter text.
<i>.key_str</i>	The <i>.key_str</i> variable contains the keys as a text

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to apply the show running-config template. pathType is a space-separated list of elements, pointing out a specific container element.
------------------------	---

Note that the tailf:cli-run-template-enter YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/runTemplateFooter

The "runTemplateFooter" element is used for specifying a template to use by the show running-config command in the C- and I-style CLIs after a set of list nodes has been displayed as a table.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to apply the show running-config template. pathType is a space-separated list of elements, pointing out a specific container element.
------------------------	---

Note that the tailf:cli-run-template-footer YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/hasRange

The "hasRange" element is used for specifying that a given non-integer key element should allow range expressions

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to allow range expressions. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the tailf:cli-allow-range YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/suppressRange

The "suppressRange" element is used for specifying that a given integer key element should not allow range expressions

Attributes:

path (pathType) The "path" attribute is mandatory. It specifies on which path to suppress range expressions. pathType is a space-separated list of elements, pointing out a specific list element.

Note that the `tailf:cli-suppress-range` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/customRange`

The "customRange" element is used for specifying that a given list element should support ranges. A type matching the range expression must be supplied, as well as a callback to use to determine if a given instance is covered by a given range expression. It contains one or more "rangeType" elements and one "callback" element.

Attributes:

path (pathType) The "path" attribute is mandatory. It specifies on which path to apply the custom range. pathType is a space-separated list of elements, pointing out a specific list element.

Note that the `tailf:cli-custom-range` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/customRange/callback`

The "callback" element is used for specifying which callback to invoke for checking if a list element instance belongs to a range. It contains a "capi" element.

Note that the `tailf:cli-custom-range-actionpoint` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/customRange/callback/capi`

The "capi" element is used for specifying the name of the callback to invoke for checking if a list element instance belongs to a range.

Attributes:

id (string) The "id" attribute is optional. It specifies a string which is passed to the callback when invoked to check if a value belongs in a range. This makes it possible to use the same callback at several locations and still keep track of which point it is invoked from.

`/clispec/$MODE/modifications/customRange/rangeType`

The "rangeType" element is used for specifying which key element of a list element should support range expressions. It is also used for specifying a matching type. All range expressions must belong to the specified type, and a valid key element must not be a valid element of this type.

Attributes:

key (string) The "key" attribute is mandatory. It specifies which key element of the list that the rangeType applies to.

<i>namespace</i> (string)	The "namespace" attribute is mandatory. It specifies which namespace the type belongs to.
<i>name</i> (string)	The "name" attribute is mandatory. It specifies the name of the range type.

Note that the `tailf:cli-range-type` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/allowWildcard`

The "allowWildcard" element is used for specifying that a given list element should allow wildcard expressions in the show pattern

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to allow wildcard expressions. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	--

Note that the `tailf:cli-allow-wildcard` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/suppressWildcard`

The "suppressWildcard" element is used for specifying that a given list element should not allow wildcard expressions in the show pattern

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to suppress wildcard expressions. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the `tailf:cli-suppress-wildcard` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/suppressValidationWarningPrompt`

The "suppressValidationWarningPrompt" element is used for specifying that for a given path a validate warning should not result in a prompt to the user. The warning is displayed but without blocking the commit operation.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies on which path to suppress the validation warning prompt. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	--

Note that the `tailf:cli-suppress-validate-warning-prompt` YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/errorMessageRewrite`

The "errorMessageRewrite" element is used for specifying that a callback should be invoked for possibly rewriting error messages before displaying them.

`/clispec/$MODE/modifications/errorMessageRewrite/callback`

The "callback" element is used for specifying which callback to invoke for rewriting a message. It contains a "capi" element.

`/clispec/$MODE/modifications/errorMessageRewrite/callback/capi`

The "capi" element is used for specifying the name of the callback to invoke for rewriting a message.

`/clispec/$MODE/modifications/showPathRewrite`

The "showPathRewrite" element is used for specifying that a callback should be invoked for possibly rewriting the show path before executing a show command. The callback is invoked by the builtin show command.

`/clispec/$MODE/modifications/showPathRewrite/callback`

The "callback" element is used for specifying which callback to invoke for rewriting the show path. It contains a "capi" element.

`/clispec/$MODE/modifications/showPathRewrite/callback/capi`

The "capi" element is used for specifying the name of the callback to invoke for rewriting the show path.

`/clispec/$MODE/modifications/noKeyCompletion`

The "noKeyCompletion" element tells the CLI to not perform completion for key elements for a given path. This is to avoid querying the data provider for all existing keys.

Attributes:

<i>src</i> (pathType)	The "src" attribute is mandatory. It specifies which path to make not do completion for. pathType is a space-separated list of elements, pointing out a specific list element.
-----------------------	--

Note that the `tailf:cli-no-key-completion` extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/noMatchCompletion`

The "noMatchCompletion" element tells the CLI to not provide match completion for a given element path for show commands.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to make not do match completion for. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the `tailf:cli-no-match-completion` YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/suppressShowMatch

The "suppressShowMatch" element makes it possible to specify that a specific completion match (ie a filter match that appear at list element nodes as an alternative to specifying a single instance) to the show command should not be available.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to suppress. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the tailf:cli-suppress-show-match YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/enforceTable

The "enforceTable" element makes it possible to force the generation of a table for a list element node regardless of whether the table will be too wide or not. This applies to the tables generated by the auto-rendered show commands for config="false" data in the C- and I- style CLIs.

Attributes:

<i>src</i> (pathType)	The "src" attribute is mandatory. It specifies which path to enforce. pathType is a space-separated list of elements, pointing out a specific list element.
-----------------------	---

Note that the tailf:cli-enforce-table YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/preformatted

The "preformatted" element makes it possible to suppress quoting of stats elements when displaying them. Newlines will be preserved in strings etc

Attributes:

<i>src</i> (pathType)	The "src" attribute is mandatory. It specifies which path to consider preformatted. pathType is a space-separated list of elements, pointing out a specific list element.
-----------------------	---

Note that the tailf:cli-preformatted YANG extension can be used to the same effect directly in YANG file.

/clispec/\$MODE/modifications/exposeKeyName

The "exposeKeyName" element makes it possible to force the C- and I-style CLIs to expose the key name to the CLI user. The user will be required to enter the name of the key and the key name will be displayed when showing the configuration.

Attributes:

<i>path</i> (pathType)	The "src" attribute is mandatory. It specifies which leaf to expose. pathType is a space-separated list of elements, pointing out a specific list key element.
------------------------	--

Note that the `tailf:cli-expose-key-name` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/displayEmptyConfig
```

The "displayEmptyConfig" element makes it possible to tell `confd` to display empty configuration list elements when displaying stats data in J-style CLI, provided that the list element has at least one optional `config="false"` element.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to apply the mod to. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the `tailf:cli-display-empty-config` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/suppressKeyAbbrev
```

The "suppressKeyAbbrev" element makes it possible to suppress the use of abbreviations for specific key elements.

Attributes:

<i>src</i> (pathType)	The "src" attribute is mandatory. It specifies which path to suppress. pathType is a space-separated list of elements, pointing out a specific list element.
-----------------------	--

Note that the `tailf:cli-suppress-key-abbreviation` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/allowKeyAbbrev
```

The "allowKeyAbbrev" element makes it possible to allow the use of abbreviations for specific key elements.

Attributes:

<i>src</i> (pathType)	The "src" attribute is mandatory. It specifies which path to suppress. pathType is a space-separated list of elements, pointing out a specific list element.
-----------------------	--

Note that the `tailf:allow-key-abbreviation` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/modeName/fixed (xs:string)
```

Specifies a fixed mode name.

Note that the `tailf:cli-mode-name` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/modeName/capi
```

Specifies that the mode name should be calculated through a callback function. It contains exactly one "cmdpoint" element.

Note that the `tailf:cli-mode-name-actionpoint` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/modeName/capi/cmdpoint (xs:string)
```

Specifies the callpoint name of the mode name function.

```
/clispec/$MODE/modifications/autocommitDelay
```

The "autocommitDelay" element makes it possible to enable transactions while in a specific submode (or submode of that mode). The modifications performed in that mode will not take effect until the user exits that submode.

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to delay autocommit for. pathType is a space-separated list of elements, pointing out a specific non-list, non-leaf element.
------------------------	---

Note that the `tailf:cli-delayed-auto-commit` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/suppressKeySort
```

The "suppressKeySort" element makes it possible to suppress sorting of key-values in the completion list. Instead the values will be displayed in the same order as they are provided by the data-provider (external or CDB).

Attributes:

<i>path</i> (pathType)	The "path" attribute is mandatory. It specifies which path to not sort. pathType is a space-separated list of elements, pointing out a specific list element.
------------------------	---

Note that the `tailf:cli-suppress-key-sort` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/legend (xs:string)
```

The "legend" element makes it possible to add a custom legend to be displayed when before printing a table. The legend is specified as a template string.

Attributes:

<i>path</i> (cmdpathType)	The "path" attribute is mandatory. It specifies for which path the legend should be printed. cmdpathType is a space-separated list of commands.
---------------------------	---

Note that the `tailf:cli-legend` YANG extension can be used to the same effect directly in YANG file.

```
/clispec/$MODE/modifications/footer (xs:string)
```

The "footer" element makes it possible to specify a template that will be displayed after printing a table.

Attributes:

<i>path</i> (cmdpathType)	The "path" attribute is mandatory. It specifies for which path the footer should be printed. cmdpathType is a space-separated list of commands.
---------------------------	---

Note that the tailf:cli-footer YANG extension can be used to the same effect directly in YANG file.

`/clispec/$MODE/modifications/help (xs:string)`

The "help" element makes it possible to add a custom help text to the specified built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to add the text to. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	--

`/clispec/$MODE/modifications/paramhelp (xs:string)`

The "paramhelp" element makes it possible to add a custom help text to a parameter to a specified built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to add the text to. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
<i>nr</i> (positiveInteger)	The "nr" attribute is mandatory. It specifies which parameter of the command to add the text to.

`/clispec/$MODE/modifications/typehelp (xs:string)`

The "typehelp" element makes it possible to add a custom help text for the built-in primitive types, e.g. to change the default type name in the CLI. For example, to display "<integer>" instead of "<unsignedShort>".

The built-in primitive types are: string, atom, normalizedString, boolean, float, decimal, double, hexBinary, base64Binary, anyURI, anySimpleType, QName, NOTATION, token, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, positiveInteger, unsignedInt, unsignedShort, unsignedByte, dateTime, date, gYearMonth, gDay, gYear, time, gMonthDay, gMonth, duration, inetAddress, inetAddressIPv4, inetAddressIP, inetAddressIPv6, inetAddressDNS, inetPortNumber, size, MD5DigestString, DES3CBCEncryptedString, AESCFB128EncryptedString, objectRef, bits_type_32, bits_type_64, hexValue, hexList, octetList, Gauge32, Counter32, Counter64, and oid.

Attributes:

<i>type</i> (xs:Name)	The "type" attribute is mandatory. It specifies which primitive type to modify.
-----------------------	---

/clispec/\$MODE/modifications/info (xs:string)

The "info" element makes it possible to add a custom info text to the specified built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to hide. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	---

/clispec/\$MODE/modifications/paraminfo (xs:string)

The "paraminfo" element makes it possible to add a custom info text to a parameter to a specified built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to add the text to. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
<i>nr</i> (positiveInteger)	The "nr" attribute is mandatory. It specifies which parameter of the command to add the text to.

/clispec/\$MODE/modifications/timeout (xs:integer|infinity)

The "timeout" element makes it possible to add a custom command timeout (in seconds) to the specified built-in command.

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to add the timeout to. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	---

/clispec/\$MODE/modifications/hide

The "hide" element makes it possible to hide a built-in command

Attributes:

<i>src</i> (cmdpathType)	The "src" attribute is mandatory. It specifies which command to hide. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.
--------------------------	---

An example:

```
<modifications>
<hide src="file show"/>
</modifications>
```

/clispec/\$MODE/modifications/hideGroup

The "hideGroup" element makes it possible to hide a built-in command under a hide group.

Attributes:

src (cmdpathType) The "src" attribute is mandatory. It specifies which command to hide. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

name (xs:string) The "name" attribute is mandatory. It specifies which hide group to hide the command.

An example:

```
<modifications>
  <hideGroup src="file show" name="debug" />
</modifications>
```

/clispec/\$MODE/modifications/submodeCommand

The "submodeCommand" element makes it possible to make a command visible in the completion lists of all submodes.

Attributes:

src (cmdpathType) The "src" attribute is mandatory. It specifies which command to make available. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

An example:

```
<modifications>
  <submodeCommand src="clear" />
</modifications>
```

/clispec/\$MODE/modifications/confirmText (xs:string)

The "confirmText" element makes it possible to add a confirmation text to the specified command, i.e. the CLI user is prompted whenever this command is executed. The prompt to be used is given as a body to the element as seen in ncs-light.cli above. The valid answers are "yes" and "no" - the text " [yes, no]" will automatically be added to the given confirmation text.

Attributes:

src (cmdpathType) The "src" attribute is mandatory. It specifies which command to add a confirmation prompt to. cmdpathType is a space-separated list of commands, pointing out a specific sub-command.

defaultOption (yes|no) The "defaultOption" attribute is optional. It makes it possible to customize if "yes" or "no" should be the default option, i.e. if the user just hits ENTER. If this element is not defined it defaults to whatever is specified by the /clispec/\$MODE/modifications/defaultConfirmOption element.

/clispec/\$MODE/modifications/defaultConfirmOption (yes|no)

The "defaultConfirmOption" element makes it possible to customize if "yes" or "no" should be the default option, i.e. if the user just hits ENTER, for the confirmation text added by the "confirmText" element.

If this element is not defined it defaults to "yes".

This element affects both `/clispec/$MODE/modifications/confirmText` and `/clispec/$MODE/cmd/confirmText` if they have not defined their "defaultOption" attributes.

`/clispec/$MODE/modifications/keymap`

The "keymap" element makes it possible to modify the key bindings in the command line editor.

Attributes:

key (xs:string)

The "key" attribute is mandatory. It specifies which sequence of keystrokes to modify.

action (keymapActionType)

The "action" attribute is mandatory. It specifies what should happen when the specified key sequence is executed. Possible values are: "unset", "new", "exist", "start_of_line", "back", "abort", "tab", "delete_forward", "delete_forward_no_eof", "end_of_line", "forward", "kill_rest", "redraw", "redraw_clear", "newline", "insert(chars)", "history_next", "history_prev", "isearch_back", "transpose", "kill_line", "quote", "word_delete_back", "yank", "end_mode", "delete", "word_delete_forward", "beginning_of_line", "delete", "end_of_line", "word_forward", "word_back", "end_of_line", "beginning_of_line", "word_back", "word_forward", "word_capitalize", "word_lowercase", "word_uppercase", "word_delete_back", "word_delete_forward", "multiline_mode", "yank_killring", and "quot". To remove a default binding use the action "remove_binding".

The default keymap is:

```
<keymap key="\^A" action="start_of_line"/>
<keymap key="\^B" action="back"/>
<keymap key="\^C" action="abort"/>
<keymap key="\^D" action="delete_forward"/>
<keymap key="\^E" action="end_of_line"/>
<keymap key="\^F" action="forward"/>
<keymap key="\^J" action="newline"/>
<keymap key="\^K" action="kill_rest"/>
<keymap key="\^L" action="redraw_clear"/>
<keymap key="\^M" action="newline"/>
<keymap key="\^N" action="history_next"/>
<keymap key="\^P" action="history_prev"/>
<keymap key="\^R" action="isearch_back"/>
<keymap key="\^T" action="transpose"/>
<keymap key="\^U" action="kill_line"/>
<keymap key="\^V" action="quote"/>
<keymap key="\^W" action="word_delete_back"/>
<keymap key="\^X" action="kill_line"/>
<keymap key="\^Y" action="yank"/>
<keymap key="\^Z" action="end_mode"/>
<keymap key="\d" action="delete"/>
<keymap key="\t" action="tab"/>
<keymap key="\b" action="delete"/>
```

```

<keymap key="\ed" action="word_delete_forward"/>
<keymap key="\e[Z" action="tab"/>
<keymap key="\e[A" action="history_prev"/>
<keymap key="\e[1~" action="beginning_of_line"/>
<keymap key="\e[3~" action="delete"/>
<keymap key="\e[4~" action="end_of_line"/>
<keymap key="\eOA" action="history_prev"/>
<keymap key="\eOB" action="history_next"/>
<keymap key="\eOC" action="forward"/>
<keymap key="\eOD" action="back"/>
<keymap key="\eOM" action="newline"/>
<keymap key="\eOp" action="insert(0)"/>
<keymap key="\eOq" action="insert(1)"/>
<keymap key="\eOr" action="insert(2)"/>
<keymap key="\eOs" action="insert(3)"/>
<keymap key="\eOt" action="insert(4)"/>
<keymap key="\eOu" action="insert(5)"/>
<keymap key="\eOv" action="insert(6)"/>
<keymap key="\eOw" action="insert(7)"/>
<keymap key="\eOx" action="insert(8)"/>
<keymap key="\eOy" action="insert(9)"/>
<keymap key="\eOm" action="insert(-)"/>
<keymap key="\eOl" action="insert(*)"/>
<keymap key="\eOn" action="insert(.)"/>
<keymap key="\e[5C" action="word_forward"/>
<keymap key="\e[5D" action="word_back"/>
<keymap key="\e[1;5C" action="word_forward"/>
<keymap key="\e[1;5D" action="word_back"/>
<keymap key="\e[B" action="history_next"/>
<keymap key="\e[C" action="forward"/>
<keymap key="\e[D" action="back"/>
<keymap key="\e[F" action="end_of_line"/>
<keymap key="\e[H" action="beginning_of_line"/>
<keymap key="\eb" action="word_back"/>
<keymap key="\ef" action="word_forward"/>
<keymap key="\ec" action="word_capitalize"/>
<keymap key="\el" action="word_lowercase"/>
<keymap key="\eu" action="word_uppercase"/>
<keymap key="\eb" action="word_delete_back"/>
<keymap key="\ed" action="word_delete_forward"/>
<keymap key="\em" action="multiline_mode"/>
<keymap key="\ey" action="yank_killring"/>
<keymap key="\eq" action="quote"/>

```

The default keymap for I-style differs with the following mapping:

```
<keymap key="^D" action="delete_forward_no_eof"/>
```

/clispec/\$MODE/show/callback/capi

The "capi" element specifies that the command is implemented using Java API using the same API as for actions. It contains one "cmdpoint" element and one or zero "args" element.

An example:

```

<callback>
  <capi>
    <cmdpoint>adduser</cmdpoint>
  </capi>

```

```
</callback>
```

```
/clispec/$MODE/show/callback/capi/args (argsType)
```

The "args" element specifies the arguments to use when executing the command specified by the "callpoint" element. argsType is a space-separated list of argument strings.

The string may contain a number of built-in variables which are expanded on execution. The built-in variables are: "cwd", "user", "groups", "ip", "maapi", "uid", "gid", "tty", "ssh_connection", "opaque", "path", "cpath", "ipath" and "licounter". In addition the variables "spath" and "ispath" are available when a command is executed from a show path. For example:

```
<args>$(user)</args>
```

Will expand to the username.

```
/clispec/$MODE/show/callback/capi/cmdpoint (xs:NCName)
```

The "cmdpoint" element specifies the name of the Java API action to be called. For this to work, a actionpoint must be registered with the NSO daemon at startup.

```
/clispec/$MODE/show/callback/exec
```

The "exec" element specifies how the command is implemented using an executable or a shell script. It contains (in order) one "osCommand" element, zero or one "args" elements and zero or one "options" elements.

An example:

```
<callback>
  <exec>
    <osCommand>cp</osCommand>
    <options>
      <uid><phrase condition="confd">confd</phrase><phrase condition="ncs">ncs</phrase></
      <wd>/var/tmp</wd>
      ...
    </options>
  </exec>
</callback>
```

```
/clispec/$MODE/show/callback/exec/osCommand (xs:token)
```

The "osCommand" element specifies the path to the executable or shell script to be called. If the command is in the \$PATH (as specified when we start the NSO daemon) the path may just be the name of the command.

The "osCommand" and "args" for "show" differs a bit from the ones for "cmd". For "show" there are a few built-in arguments that always are given to the "osCommand". These are appended to "args". The built-in arguments are "0", the keypath (ispath) and an optional filter. Like this: "0 /prefix:keypath *".

The command is not paginated by default in the CLI and will only do so if it is piped to more.

```
joe@io> example_os_command | more
```

The command is invoked as if it had been executed by exec(3), i.e. not in a shell environment such as "/bin/sh -c ...".

/clispec/\$MODE/show/callback/exec/args (argsType)

The "args" element specifies additional arguments to use when executing the command specified by the "osCommand" element. The "args" arguments are prepended to the mandatory ones listed in "osCommand". argsType is a space-separated list of argument strings.

The string may contain a number of built-in variables which are expanded on execution. The built-in variables are: "cwd", "user", "groups", "ip", "maapi", "uid", "gid", "tty", "ssh_connection", "opaque", "path", "cpath", "ipath" and "licounter". In addition the variables "spath" and "ispath" are available when a command is executed from a show path. For example:

```
<args>$(user) </args>
```

Will expand to the username and the three built-in arguments. For example: "admin 0 / prefix:keypath *".

/clispec/\$MODE/show/callback/exec/options

The "options" element specifies how the command is be executed. It contains (in any order) zero or one "uid" elements, zero or one "gid" elements, zero or one "wd" elements, zero or one "batch" elements, zero or one "pty" element, zero or one of "interrupt" elements, zero or one of "noInput", and zero or one "ignoreExitValue" elements.

/clispec/\$MODE/show/callback/exec/options/uid (idType) [confd]

The "uid" element specifies which user id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same user id as the NSO daemon.
<i>user</i>	The command is run as the same user id as the user logged in to the CLI, i.e. we have to make sure that this user id exists as an actual user id on the device.
<i>root</i>	The command is run as root.
<i><uid></i> (the numerical user <i><uid></i>)	The command is run as the user id <i><uid></i> . <i>Note:</i> If uid is set to either "user", "root" or "<uid>" the the NSO daemon must have been started as root (or setuid), or the showptywrapper must have setuid root permissions.

/clispec/\$MODE/show/callback/exec/options/gid (idType) [confd]

The "gid" element specifies which group id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same group id as the NSO daemon.
<i>user</i>	The command is run as the same group id as the user logged in to the CLI, i.e. we have to make sure that this group id exists as an actual group on the device.

<i>root</i>	The command is run as root.
<i><gid></i> (the numerical group <i><gid></i>)	The command is run as the group id <i><gid></i> . <i>Note:</i> If gid is set to either "user", "root" or " <i><gid></i> " the the NSO daemon must have been started as root (or <code>setuid</code>), or the <code>showptywrapper</code> must have <code>setuid</code> root permissions.

`/clispec/$MODE/show/callback/exec/options/wd (xs:token)`

The "wd" element specifies which working directory to use when executing the command. If not given, the command is executed from the location of the CLI.

`/clispec/$MODE/show/callback/exec/options/pty (xs:boolean)`

The "pty" element specifies weather a pty should be allocated when executing the command. The default is to allocate a pty for operational and configure `osCommands`, but not for `osCommands` executing as a pipe command. This behavior can be overridden with this parameter.

`/clispec/$MODE/show/callback/exec/options/interrupt (interruptType) [sigkill]`

The "interrupt" element specifies what should happen when the user enters `ctrl-c` in the CLI. Possible values are:

<i>sigkill</i> (default)	The command is terminated by sending the <code>sigkill</code> signal.
<i>sigint</i>	The command is interrupted by the <code>sigint</code> signal.
<i>sigterm</i>	The command is interrupted by the <code>sigterm</code> signal.
<i>ctrlc</i>	The command is sent the <code>ctrl-c</code> character which is interpreted by the pty.

`/clispec/$MODE/show/callback/exec/options/ignoreExitValue`

The "ignoreExitValue" element specifies that the CLI engine should ignore the fact that the command returns a non-zero value. Normally it signals an error on `stdout` if a non-zero value is returned.

`/clispec/$MODE/show/callback/exec/options/globalNoDuplicate (xs:token)`

The "globalNoDuplicate" element specifies that only one instance with the same name can be run at any one time in the system. The command can be started either from the CLI, the Web UI or through `NETCONF`.

`/clispec/$MODE/show/callback/exec/options/noInput (xs:token)`

The "noInput" element specifies that the command should not grab the input stream and consume freely from that. This option should be used if the command should not consume input characters. If not used then the command will eath all data from the input stream and cut-and-paste may not work as intended.

`/clispec/$MODE/show/options`

The "options" element specifies under what circumstances the CLI command should execute. It contains (in any order) zero or one "notInterruptible" elements, zero or one of "displayWhen" elements, and zero or one "paginate" elements.

`/clispec/$MODE/show/options/notInterruptible`

The "notInterruptible" element disables <ctrl-c> and the execution of the CLI command can thus not be interrupted.

/clispec/\$MODE/show/options/paginate

The "paginate" element enables a filter for paging through CLI command output text one screen at a time.

/clispec/\$MODE/show/options/displayWhen

The "displayWhen" element can be used to add a displayWhen xpath condition to a command.

Attributes:

expr (xpath expression)

The "expr" attribute is mandatory. It specifies an xpath expression. If the expression evaluates to true then the command is available, otherwise not.

ctx (path)

The "ctx" attribute is optional. If not specified the current editpath/mode-path is used as context node for the xpath evaluation. Note that the xpath expression will automatically evaluate to false if a display when expression is used for a top-level command and no ctx is specified. The path may contain variables defined in the dict.

/clispec/operationalMode/start

The "start" command is executed when the CLI is started. It can be used to, for example, remind the user to change an expired password. It contains (in order) zero or one "callback" elements, and zero or one "options" elements.

This element must occur after the <modifications> section and before any <cmd> entries.

An example:

```
<start>
  <callback>
    <exec>
      <osCommand>./startup.sh</osCommand>
    </exec>
  </callback>
</start>
```

/clispec/operationalMode/start/callback

The "callback" element specifies how the command is implemented, e.g. as a OS executable or an API callback. It contains one of the elements "capi", and "exec".

/clispec/operationalMode/start/callback/capi

The "capi" element specifies that the command is implemented using Java API using the same API as for actions. It contains one "cmdpoint" element.

An example:

```
<callback>
```

```

    <capi>
      <cmdpoint>adduser</cmdpoint>
    </capi>
  </callback>

```

/clispec/operationalMode/start/callback/capi/cmdpoint (xs:NCName)

The "cmdpoint" element specifies the name of the Java API action to be called. For this to work, a actionpoint must be registered with the NSO daemon at startup.

/clispec/operationalMode/start/callback/exec

The "exec" element specifies how the command is implemented using an executable or a shell script. It contains (in order) one "osCommand" element, zero or one "args" elements and zero or one "options" elements.

An example:

```

<callback>
  <exec>
    <osCommand>cp</osCommand>
    <options>
      <uid>confd</uid>
      <wd>/var/tmp</wd>
      ...
    </options>
  </exec>
</callback>

```

/clispec/operationalMode/start/callback/exec/osCommand (xs:token)

The "osCommand" element specifies the path to the executable or shell script to be called. If the command is in the \$PATH (as specified when we start the NSO daemon) the path may just be the name of the command.

The command is invoked as if it had been executed by exec(3), i.e. not in a shell environment such as "/bin/sh -c ...".

/clispec/operationalMode/start/callback/exec/args (argsType)

The "args" element specifies the arguments to use when executing the command specified by the "osCommand" element. argsType is a space-separated list of argument strings. The built-in variables are: "cwd", "user", "groups", "ip", "maapi", "uid", "gid", "tty", "ssh_connection", "opaque", "path", "cpath", "ipath" and "licounter". In addition the variables "spath" and "ispath" are available when a command is executed from a show path. For example:

```
<args>$(user)</args>
```

Will expand to the username.

/clispec/operationalMode/start/callback/exec/options

The "options" element specifies how the command is be executed. It contains (in any order) zero or one "uid" elements, zero or one "gid" elements, zero or one "wd" elements, zero or one "batch" elements, zero or one of "interrupt" elements, and zero or one "ignoreExitValue" elements.

/clispec/operationalMode/start/callback/exec/options/uid (idType) [confd]

The "uid" element specifies which user id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same user id as the NSO daemon.
<i>user</i>	The command is run as the same user id as the user logged in to the CLI, i.e. we have to make sure that this user id exists as an actual user id on the device.
<i>root</i>	The command is run as root.
<i><uid></i> (the numerical user <i><uid></i>)	The command is run as the user id <i><uid></i> . <i>Note:</i> If uid is set to either "user", "root" or " <i><uid></i> " the the NSO daemon must have been started as root (or <i>setuid</i>), or the <i>startptywrapper</i> must have <i>setuid</i> root permissions.

`/clispec/operationalMode/start/callback/exec/options/gid (idType) [confd]`

The "gid" element specifies which group id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same group id as the NSO daemon.
<i>user</i>	The command is run as the same group id as the user logged in to the CLI, i.e. we have to make sure that this group id exists as an actual group on the device.
<i>root</i>	The command is run as root.
<i><gid></i> (the numerical group <i><gid></i>)	The command is run as the group id <i><gid></i> . <i>Note:</i> If gid is set to either "user", "root" or " <i><gid></i> " the the NSO daemon must have been started as root (or <i>setuid</i>), or the <i>startptywrapper</i> must have <i>setuid</i> root permissions.

`/clispec/operationalMode/start/callback/exec/options/wd (xs:token)`

The "wd" element specifies which working directory to use when executing the command. If not given, the command is executed from the location of the CLI.

`/clispec/operationalMode/start/callback/exec/options/globalNoDuplicate (xs:token)`

The "globalNoDuplicate" element specifies that only one instance with the same name can be run at any one time in the system. The command can be started either from the CLI, the Web UI or through NETCONF.

`/clispec/operationalMode/start/callback/exec/options/interrupt (interruptType) [sigkill]`

The "interrupt" element specifies what should happen when the user enters ctrl-c in the CLI. Possible values are:

<i>sigkill</i> (default)	The command is terminated by sending the <i>sigkill</i> signal.
--------------------------	---

<i>sigint</i>	The command is interrupted by the sigint signal.
<i>sigterm</i>	The command is interrupted by the sigterm signal.
<i>ctrlc</i>	The command is sent the ctrl-c character which is interpreted by the pty.

```
/clispec/operationalMode/start/callback/exec/options/
ignoreExitValue(xs:boolean) [false]
```

The "ignoreExitValue" element specifies if the CLI engine should ignore the fact that the command returns a non-zero value. Normally it signals an error on stdout if a non-zero value is returned.

```
/clispec/operationalMode/start/options
```

The "options" element specifies under what circumstances the CLI command should execute. It contains (in any order) zero or one "notInterruptible" elements, and zero or one "paginate" elements.

```
/clispec/operationalMode/start/options/notInterruptible
```

The "notInterruptible" element disables <ctrl-c> and the execution of the CLI command can thus not be interrupted.

```
/clispec/operationalMode/start/options/paginate
```

The "paginate" element enables a filter for paging through CLI command output text one screen at a time.

```
/clispec/$MODE/cmd
```

The "cmd" element adds a new command to the CLI hierarchy as defined by its "mount" and "mode" attributes. It contains (in order) one "info" element, one "help" element, zero or one "confirmText" element, zero or one "callback" elements, zero or one "params" elements, zero or one "options" elements and finally zero or more "cmd" elements (recursively).

Attributes:

<i>name</i> (xs:NCName)	The "name" attribute is mandatory. It specifies the name of the command.
<i>extend</i> (xs:boolean) [false]	The "extend" attribute is optional. It specifies that the command should be mounted on top of an existing command, ie with the exact same name as an existing command but with different parameters. Which command is executed depends on which parameters are supplied when the command is invoked. This can be used to overlay an existing command.
<i>mount</i> (cmdpathType) []	The "mount" attribute is optional. It specifies where in the command hierarchy of built-in commands this command should be mounted. If no mount attribute is given, or if it is empty (""), the command is mounted on the top-level of the CLI hierarchy.

An example:

```

<cmd name="copy" mount="file">
  <info>Copy a file</info>
  <help>Copy a file from in the file system.</help>
  <callback>
    <exec>
      <osCommand>cp</osCommand>
      <options>
        <uid>confd</uid>
      </options>
    </exec>
  </callback>
  <params>
    <param>
      <type><file/></type>
      <info>&lt;source file&gt;</info>
    </param>
    <param>
      <type><file/></type>
      <info>&lt;destination&gt;</info>
    </param>
  </params>

  <cmd ...>
    ...
  </cmd>

  <cmd ...>
    ...
  </cmd>
</cmd>

```

```
/clispec/$MODE/cmd/info (xs:string)
```

The "info" element is a single text line describing the command.

An example:

```

<cmd name="start">
  <info>Start displaying the system log or trace a file</info>
  ...

```

and when we do the following in the CLI we get:

```

joe@xev> monitor st<TAB>
Possible completions:
  start - Start displaying the system log or trace a file
  stop  - Stop displaying the system log or trace a file
joe@xev> monitor st

```

```
/clispec/$MODE/cmd/help (xs:string)
```

The "help" element is a multi-line text string describing the command. This text is shown when we use the "help" command.

An example:

```

joe@xev> help monitor start
Help for command: monitor start
Start displaying the system log or trace a file in the background.

```

We can abort the logging using the "monitor stop" command.
joe@xev>

/clispec/\$MODE/cmd/timeout (xs:integer|infinity)

The "timeout" element is a timeout for the command in seconds. Default is infinity.

/clispec/\$MODE/cmd/confirmText

See /clispec/\$MODE/modifications/confirmText

/clispec/\$MODE/cmd/callback

The "callback" element specifies how the command is implemented, e.g. as a OS executable or a CAPI callback. It contains one of the elements "capi", "exec", "table" or "execStop".

Note: A command which has a callback defined may not have recursive sub-commands. Likewise, a command which has recursive sub-commands may not have a callback defined. A command without sub-commands must have a callback defined.

/clispec/\$MODE/cmd/callback/table

The "table" element specifies that the command should display parts of the configuration in the form of a table.

An example:

```
<callback>
  <table>
    <root>/all:config/hosts/host</root>
    <item>
      <width>20</width>
      <header>NAME</header>
      <path>name</path>
      <align>left</align>
    </item>
    <item>
      <header>DOMAIN</header>
      <path>domain</path>
    </item>
    <item>
      <header>IP</header>
      <path>interfaces/interface/ip</path>
      <align>right</align>
    </item>
  </table>
</callback>
```

/clispec/\$MODE/cmd/callback/table/root (xs:string)

Should be a path to a list element. All item paths in the table are relative to this path.

/clispec/\$MODE/cmd/callback/table/legend (xs:string)

Should be a legend template to display before showing the table.

/clispec/\$MODE/cmd/callback/table/footer (xs:string)

Should be a footer template to display after showing the table.

/clispec/\$MODE/cmd/callback/table/item

Specifies a column in the table. It contains a "header" element and a "path" element, and optionally a "width" element.

/clispec/\$MODE/cmd/callback/table/item/header (xs:string)

Header of this column in the table.

/clispec/\$MODE/cmd/callback/table/item/path (xs:string)

Path to the element in this column.

/clispec/\$MODE/cmd/callback/table/item/width (xs:integer)

The width in characters of this column.

/clispec/\$MODE/cmd/callback/table/item/align (left|right|center)

The data alignment of this column.

/clispec/\$MODE/cmd/callback/capi

The "capi" element specifies that the command is implemented using Java API using the same API as for actions. It contains one "cmdpoint" element.

An example:

```
<callback>
  <capi>
    <cmdpoint>adduser</cmdpoint>
  </capi>
</callback>
```

/clispec/\$MODE/cmd/callback/capi/cmdpoint (xs:NCName)

The "cmdpoint" element specifies the name of the Java API action to be called. For this to work, a actionpoint must be registered with the NSO daemon at startup.

/clispec/\$MODE/cmd/callback/exec

The "exec" element specifies how the command is implemented using an executable or a shell script. It contains (in order) one "osCommand" element, zero or one "args" elements and zero or one "options" elements.

An example:

```
<callback>
  <exec>
    <osCommand>cp</osCommand>
    <options>
      <uid>confd</uid>
      <wd>/var/tmp</wd>
      ...
    </options>
  </exec>
</callback>
```

/clispec/\$MODE/cmd/callback/exec/osCommand (xs:token)

The "osCommand" element specifies the path to the executable or shell script to be called. If the command is in the \$PATH (as specified when we start the NSO daemon) the path may just be the name of the command.

The command is invoked as if it had been executed by `exec(3)`, i.e. not in a shell environment such as `"/bin/sh -c ..."`.

`/clispec/$MODE/cmd/callback/exec/args (argsType)`

The "args" element specifies the arguments to use when executing the command specified by the "osCommand" element. `argsType` is a space-separated list of argument strings. The built-in variables are: "cwd", "user", "groups", "ip", "maapi", "uid", "gid", "tty", "ssh_connection", "opaque", "path", "cpath", "ipath" and "licounter". The variable "pipecmd_XYZ" can be used to determine whether a certain builtin pipe command has been run together with the command. Here XYZ is the name of the pipe command. An example of such a variable is "pipecmd_include". In addition the variables "spath" and "ispath" are available when a command is executed from a show path. For example:

```
<args>$(user)</args>
```

Will expand to the username.

`/clispec/$MODE/cmd/callback/exec/options`

The "options" element specifies how the command is be executed. It contains (in any order) zero or one "uid" elements, zero or one "gid" elements, zero or one "wd" elements, zero or one "batch" elements, zero or one of "interrupt" elements, and zero or one "ignoreExitValue" elements.

`/clispec/$MODE/cmd/callback/exec/options/uid (idType) [confd]`

The "uid" element specifies which user id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same user id as the NSO daemon.
<i>user</i>	The command is run as the same user id as the user logged in to the CLI, i.e. we have to make sure that this user id exists as an actual user id on the device.
<i>root</i>	The command is run as root.
<i><uid></i> (the numerical user <i><uid></i>)	The command is run as the user id <i><uid></i> . <i>Note:</i> If uid is set to either "user", "root" or " <i><uid></i> " the the NSO daemon must have been started as root (or <code>setuid</code>), or the <code>cmdptywrapper</code> must have <code>setuid</code> root permissions.

`/clispec/$MODE/cmd/callback/exec/options/gid (idType) [confd]`

The "gid" element specifies which group id to use when executing the command. Possible values are:

<i>confd</i> (default)	The command is run as the same group id as the NSO daemon.
<i>user</i>	The command is run as the same group id as the user logged in to the CLI, i.e. we have to make sure that this group id exists as an actual group on the device.
<i>root</i>	The command is run as root.

`<gid>` (the numerical group
`<gid>`)

The command is run as the group id `<gid>`.

Note: If gid is set to either "user", "root" or "`<gid>`" the NSO daemon must have been started as root (or `setuid`), or the `cmdptywrapper` must have `setuid` root permissions.

`/clispec/$MODE/cmd/callback/exec/options/wd (xs:token)`

The "wd" element specifies which working directory to use when executing the command. If not given, the command is executed from the location of the CLI.

`/clispec/$MODE/cmd/callback/exec/options/pty (xs:boolean)`

The "pty" element specifies whether a pty should be allocated when executing the command. The default is to allocate a pty for operational and configure `osCommands`, but not for `osCommands` executing as a pipe command. This behavior can be overridden with this parameter.

`/clispec/$MODE/cmd/callback/exec/options/globalNoDuplicate (xs:token)`

The "globalNoDuplicate" element specifies that only one instance with the same name can be run at any one time in the system. The command can be started either from the CLI, the Web UI or through `NETCONF`.

`/clispec/$MODE/cmd/callback/exec/options/noInput (xs:token)`

The "noInput" element specifies that the command should not grab the input stream and consume freely from that. This option should be used if the command should not consume input characters. If not used then the command will eat all data from the input stream and cut-and-paste may not work as intended.

`/clispec/$MODE/cmd/callback/exec/options/batch`

The "batch" element makes it possible to specify that a command returns immediately but still runs in the background, optionally generating output on `stdout`. An example of such a command is the standard "monitor start" command, which prints additional data appended to a (log) file:

```
joe@io> monitor start /var/log/messages
joe@io>
log: Apr 10 11:59:32 earth ntpd[530]: kernel time sync enabled 2001
```

Ten seconds later...

```
log: Apr 12 01:59:02 earth sshd[26847]: error: PAM: auth error for cathy
joe@io> monitor stop /var/log/messages
joe@io>
```

The "batch" element contains (in order) one "group" element, an optional "prefix" element, and an optional "noDuplicate" element. The prefix defaults to the empty string.

An example from `ncs.cli` implementing the monitor functionality:

```
<cmd name="start">
...
<callback>
  <exec>
    <osCommand>tail</osCommand>
    <args>-f -n 0</args>
```

```

        <options>
        ...
        <batch>
            <group>monitor_file</group>
            <prefix>log:</prefix>
            <noDuplicate/>
        </batch>
    </options>
</exec>
</callback>
...
</cmd>

```

The batch group is used to kill the command as exemplified in the "execStop" element description below. "noDuplicate" indicates that a specific file is not allowed to be monitored by several commands in parallel.

/clispec/\$MODE/cmd/callback/exec/options/batch/group (xs:NCName)

The "group" element attaches a group label to the command. The group label is used when defining a "stop" command whose job it is to kill the background command. Take a look at the monitor example above for better understanding.

The stop command is defined using a "execStop" element as described below.

/clispec/\$MODE/cmd/callback/exec/options/batch/prefix (xs:NCName)

The "prefix" element specifies a string to prepend to all lines printed by the background command. In the monitor example above, "log:" is the chosen prefix.

/clispec/\$MODE/cmd/callback/exec/options/batch/noDuplicate

The "noDuplicate" element specifies that only a single instance of this batch command, including the given/specified parameters, can run in the background.

/clispec/\$MODE/cmd/callback/exec/options/interrupt (interruptType) [sigkill]

The "interrupt" element specifies what should happen when the user enters ctrl-c in the CLI. Possible values are:

<i>sigkill</i> (default)	The command is terminated by sending the sigkill signal.
<i>sigint</i>	The command is interrupted by the sigint signal.
<i>sigterm</i>	The command is interrupted by the sigterm signal.
<i>ctrlc</i>	The command is sent the ctrl-c character which is interpreted by the pty.

/clispec/\$MODE/cmd/callback/exec/options/ignoreExitValue(xs:boolean) [false]

The "ignoreExitValue" element specifies if the CLI engine should ignore the fact that the command returns a non-zero value. Normally it signals an error on stdout if a non-zero value is returned.

/clispec/\$MODE/cmd/callback/execStop

The "execStop" element specifies that a command defined by an "exec" element is to be killed.

Attributes:

batchGroup (xs:NCName)

The "batchGroup" attribute is mandatory. It specifies a background command to kill. It corresponds to a group label defined by another "exec" command using the "batch" element.

An example from ncs.cli which kills a background monitor session:

```
<cmd name="stop">
  ...
  <callback>
    <execStop batchGroup="monitor_file"/>
  </callback>
  ...
</cmd>
```

/clispec/\$MODE/cmd/params

The "params" element lists which parameters the CLI should prompt for. These parameters are then used as arguments to either the CAPI callback or the OS executable command (as specified by the "capi" element or the "exec" element, respectively). If an "args" element as well as a "params" element has been specified, all of them are used as arguments: first the "args" arguments and then the "params" values are passed to the CAPI callback or executable.

The "params" element contains (in order) zero or more "param" elements and zero or one "any" elements.

Attributes:

mode (list|choice)

This is an optional attribute. If it is "choice" then at least "min" and at most "max" params must be given by the user. If it is "list" then all non-optional parameters must be given the command in the order they appear in the list.

min (xs:nonNegativeInteger)

This optional attribute defines the minimum number of parameters from the body of the "params" element that the user must supply with the command. It is only applicable if the mode attribute has been set to "choice". The default value is "1".

max (xs:nonNegativeInteger | unlimited)

This optional attribute defines the maximum number of parameters from the body of the "params" element that the user may supply with the command. It is only applicable if the mode attribute has been set to "choice". The default value is "1" unless multi is specified, in which case the default is "unlimited".

multi (xs:boolean)

This optional attribute controls if each parameters should be allowed to be entered more than once. If set to "true" then each parameter may occur multiple times. The default is "false".

An example from ncs.cli which copies one file to another:

```
<params>
```

```

    <param>
      <type><file/></type>
      ...
    </param>
    <param>
      <type><file/></type>
      ...
    </param>
    ...
  </params>

```

```
/clispec/$MODE/cmd/params/param
```

The "param" element defines the nature of a single parameter which the CLI should prompt for. It contains (in any order) zero or one "type" element, zero or one "info" element, zero or one "help" element, zero or one "optional" element, zero or one "name" element, zero or one "params" element, zero or one "auditLogHide" element, zero or one "prefix" element, zero or one "flag" element, zero or one "id" element, zero or one "hideGroup" element, and zero or one "simpleType" element and zero or one "completionId" element.

```
/clispec/$MODE/cmd/params/param/type
```

The "type" element is optional and defines the parameter type. It contains either a "enums", "enumerate", "void", "keypath", "file", "url_file", "simpleType", "xpath", "url_directory_file", "directory_file", "url_directory" or a "directory" element. If the "type" element is not present, the value entered by the user is passed unmodified to the callback.

```
/clispec/$MODE/cmd/params/param/type/enums (enumsType)
```

The "enums" element defines a list of allowed enum values for the parameter. enumsType is a space-separated list of string enums.

An example:

```
<enums>for bar baz</enums>
```

```
/clispec/$MODE/cmd/params/param/type/enumerate
```

The "enumerate" is used to define a set of values with info text. It can contain one of more of the element "elem".

```
/clispec/$MODE/cmd/params/param/type/enumerate/enum
```

The "enum" is used to define an enumeration value with help text. It must contain the element "name" and optionally an "info" element and a "hideGroup" element.

```
/clispec/$MODE/cmd/params/param/type/enumerate/enum/name(xs:token)
```

The "name" is used to define the name of an enumeration.

```
/clispec/$MODE/cmd/params/param/type/enumerate/enum/info(xs:string)
```

The "info" is used to define the info that is displayed during completion in the CLI. The element is optional.

```
/clispec/$MODE/cmd/params/param/type/enumerate/enum/hideGroup(xs:string)
```

The "hideGroup" element makes an enum value invisible and it cannot be used even if a user knows about its existence. The enum value will become visible when the hide group is 'unhidden' using the unhide command.

/clispec/\$MODE/cmd/params/param/type/void

The "void" element is used to indicate that this parameter should not prompt for a value. It can only be used when the "name" element is used.

/clispec/\$MODE/cmd/params/param/type/keypath (keypathType)

The "keypath" element specifies that the parameter must be a keypath pointing to a configuration value. Valid keypath values are: *new* or *exist*:

new The keypath is either an already existing configuration value or an instance value to be created.

exist The keypath must be an already existing configuration value.

/clispec/\$MODE/cmd/params/param/type/key (path)

The "key" element specifies that the parameter is an instance identifier, either an existing instance or a new. If the list has multiple key elements then they will be entered with a space in between.

The path should point to a list element, not the actual key leaf. If the list has multiple keys then they user will be requested to enter all keys of an instance. The path may be either absolute or relative to the current submode path. Also variables referring to key elements in the current submode path may be used, where the closes key is named \$(key-1-1), \$(key-1-2) etc. Eg

/foo{key-2-1,key-2-2}/bar{key-1-1,key-1-2}/...

Attributes:

mode (keypathType) The "mode" attribute is mandatory. It specifies if the parameter refers to an existing (exist) instance or a new (new) instance.

/clispec/\$MODE/cmd/params/param/type/pattern (patternType)

The "pattern" element specifies that the parameter must be a show command pattern. Valid pattern values are: *stats* or *config* or *all*:

stats The pattern is only related to "config false" nodes in the data model. Note that CLI modifications such as fullShowPath, incompleteShowPath etc are applied to this pattern.

config The pattern is only related to "config true" elements in the data model.

all The pattern spans over all visible nodes in the data model.

/clispec/\$MODE/cmd/params/param/type/file

The "file" element specifies that the parameter is a file on disk. The CLI automatically enables tab completion to help the user to choose the correct file.

Attributes:

wd (xs:token) The "wd" attribute is optional. It specifies a working directory to be used as the root for the tab completion algorithm. If no "wd" attribute is specified, the working directory is as defined for the "/clispec/\$MODE/cmd/callback/exec/options/wd" element.

An example:

```
<file wd="/var/log/" />
```

```
/clispec/$MODE/cmd/params/param/type/url_file
```

The "url_file" element specifies that the parameter is a file on disk or an URL. The CLI automatically enables tab completion to help the user to choose the correct file.

Attributes:

wd (xs:token) The "wd" attribute is optional. It specifies a working directory to be used as the root for the tab completion algorithm. If no "wd" attribute is specified, the working directory is as defined for the "/clispec/\$MODE/cmd/callback/exec/options/wd" element.

An example:

```
<file wd="/var/log/" />
```

```
/clispec/$MODE/cmd/params/param/type/directory
```

The "directory" element specifies that the parameter is a directory on disk. The CLI automatically enables tab completion to help the user choose the correct directory.

Attributes:

wd (xs:token) The "wd" attribute is optional. It specifies a working directory to be used as the root for the tab completion algorithm. If no "wd" attribute is specified, the working directory is as defined for the "wd" element.

An example:

```
<directory wd="/var/log/" />
```

```
/clispec/$MODE/cmd/params/param/type/url_directory
```

The "url_directory" element specifies that the parameter is a directory on disk or an URL. The CLI automatically enables tab completion to help the user choose the correct directory.

Attributes:

wd (xs:token) The "wd" attribute is optional. It specifies a working directory to be used as the root for the tab completion algorithm. If no "wd" attribute is specified, the working directory is as defined for the "wd" element.

An example:

```
<directory wd="/var/log/" />
```

```
/clispec/$MODE/cmd/params/param/type/directory_file
```

The "directory_file" element specifies that the parameter is a directory or a file on disk. The CLI automatically enables tab completion to help the user choose the correct directory or file.

An example:

```
<directory_file/>
```

```
/clispec/$MODE/cmd/params/param/type/url_directory_file
```

The "url_directory_file" element specifies that the parameter is a directory or a file on disk or an URL. The CLI automatically enables tab completion to help the user choose the correct directory or file.

An example:

```
<directory_file/>
```

```
/clispec/$MODE/cmd/params/param/info (xs:string)
```

The "info" element is a single text line describing the parameter.

An example:

```
<cmd name="id" mount="">
  <info>Find uid and groups of a user</info>
  <help>Find uid and groups of a user, using the id program</help>
  <callback>
    <exec>
      <osCommand>id</osCommand>
    </exec>
  </callback>
  <params>
    <param>
      <info>User name</info>
      <help>User name</help>
    </param>
  </params>
</cmd>
```

and when we do the following in the CLI we get:

```
joe@x15> id <TAB>
User name
joe@x15> id snmp
uid=108(snmp) gid=65534(nogroup) groups=65534(nogroup)
[ok][2006-08-30 14:51:28]
```

Note: This description is *only* shown if the "type" element is left out.

```
/clispec/$MODE/cmd/params/param/help (xs:string)
```

The "help" element is a multi-line text string describing the parameter. This text is shown when we use the '?' character.

```
/clispec/$MODE/cmd/params/param/hideGroup (xs:string)
```

The "hideGroup" element makes a CLI parameter invisible and it cannot be used even if a user knows about its existence. The parameter will become visible when the hide group is 'unhidden' using the unhide command.

This mechanism correspond to the 'tailf:hidden' statement in a YANG module.

/clispec/\$MODE/cmd/params/param/name (xs:token)

The "name" element is a token which has to be entered by the user before entering the actual parameter value. It is used to get named parameters.

An example:

```
<cmd name="copy" mount="file">
  <info>Copy a file</info>
  <help>Copy a file from one location to another in the file system</help>
  <callback>
    <exec>
      <osCommand>cp</osCommand>
      <options>
        <uid>user</uid>
      </options>
    </exec>
  </callback>
  <params>
    <param>
      <type><file/></type>
      <info>&lt;source file&gt;</info>
      <help>source file</help>
      <name>from</name>
    </param>
    <param>
      <type><file/></type>
      <info>&lt;destination file&gt;</info>
      <help>destination file</help>
      <name>to</name>
    </param>
  </params>
</cmd>
```

The result is that the user has to enter

```
file copy from /tmp/orig to /tmp/copy
```

/clispec/\$MODE/cmd/params/param/prefix (xs:string)

The "prefix" element is a string that is prepended to the argument before calling the osCommand. This can be used to add Unix style command flags in front of the supplied parameters.

An example:

```
<cmd name="ssh">
  <info>Open a secure shell on another host</info>
  <help>Open a secure shell on another host</help>
  <callback>
    <exec>
      <osCommand>ssh</osCommand>
      <options>
        <uid>user</uid>
        <interrupt>ctrlc</interrupt>
      </options>
    </exec>
  </callback>
  <params>
    <param>
      <info>&lt;login&gt;</info>
      <help>Users login name on host</help>
```

```

        <name>user</name>
        <prefix>--login=</prefix>
    </param>
    <param>
        <info>&lt;host&gt;</info>
        <help>host name or IP</help>
        <name>host</name>
    </param>
</params>
</cmd>

```

The user would enter for example

```
ssh user joe host router.intranet.net
```

and the resulting call to the ssh executable would become

```
ssh --login=joe router.intranet.net
```

```
/clispec/$MODE/cmd/params/param/flag (xs:string)
```

The "flag" element is a string that is prepended to the argument before calling the osCommand. In contrast to the prefix element it will not be appended to the current parameter, but instead appear as a separate argument, ie instead of adding a unix style flag as "--foo=" (prefix) you add arguments in the style of "-f <param>" where -f is one arg and <param> is another. Both <prefix> and <flag> can be used at the same time.

An example:

```

<cmd name="ssh">
    <info>Open a secure shell on another host</info>
    <help>Open a secure shell on another host</help>
    <callback>
        <exec>
            <osCommand>ssh</osCommand>
            <options>
                <uid>user</uid>
                <interrupt>ctrlc</interrupt>
            </options>
        </exec>
    </callback>
    <params>
        <param>
            <info>&lt;login&gt;</info>
            <help>Users login name on host</help>
            <name>user</name>
            <flag>-l</flag>
        </param>
        <param>
            <info>&lt;host&gt;</info>
            <help>host name or IP</help>
            <name>host</name>
        </param>
    </params>
</cmd>

```

The user would enter for example

```
ssh user joe host router.intranet.net
```

and the resulting call to the ssh executable would become

```
ssh -l joe router.intranet.net
```

/clispec/\$MODE/cmd/params/param/id (xs:string)

The "id" is used for identifying the value of the parameter and can be used as a variable in the value of a key parameter.

An example:

```
<cmd name="test">
  <info/>
  <help/>
  <callback>
    <exec>
      <osCommand>/bin/echo</osCommand>
    </exec>
  </callback>
  <params>
    <param>
      <name>host</name>
      <id>h</id>
      <type><key mode="exist">/host</key></type>
    </param>
    <param>
      <name>interface</name>
      <type><key mode="exist">/host{$(h)}/interface</key></type>
    </param>
  </params>
</cmd>
```

There are also three builtin variables: user, uid and gid. The id and the builtin variables can be used in when specifying the path value of a key parameter, and also when specifying the wd attribute of the file, url_file, directory, and url_directory.

/clispec/\$MODE/cmd/params/param/callback/capi

Specifies that the parameter completion should be calculated through a callback function. It contains exactly one "completionpoint" element.

/clispec/\$MODE/cmd/params/param/auditLogHide

The "auditLogHide" element specifies that the parameter should be obfuscated in the audit log. This is suitable when clear text passwords are passed as command parameters.

/clispec/\$MODE/cmd/params/param/optional

The "optional" element specifies that the parameter is optional and not required. It contains zero or one "default" element. It cannot be used inside a params of type "choice".

/clispec/\$MODE/cmd/params/param/optional/default

The "default" element makes it possible to specify a default value, should the parameter be left out.

An example:

```
<optional>
  <default>42</default>
</optional>
```

/clispec/\$MODE/cmd/params/any

The "any" element specifies that any number of parameters are allowed. It contains (in any order) one "info" element and one "help" element.

```
/clispec/$MODE/cmd/params/any/info (xs:string)
```

The "info" element is a single text line describing the parameter(s) expected.

An example:

```
<cmd name="evaluate" mount="">
  <info>Evaluate an arithmetic expression</info>
  <help>Evaluate an arithmetic expression, using the expr program</help>
  <callback>
    <exec>
      <osCommand>expr</osCommand>
    </exec>
  </callback>
  <params>
    <any>
      <info>Arithmetic expression</info>
      <help>Arithmetic expression</help>
    </any>
  </params>
</cmd>
```

and when we do the following in the CLI we get:

```
joe@xev> eva<TAB>
joe@xev> evaluate <TAB>
Arithmetic expression
joe@xev> evaluate 2 + 5
7
[ok][2006-08-30 14:47:17]
```

```
/clispec/$MODE/cmd/params/any/help (xs:string)
```

The "help" element is a multi-line text string describing these anonymous parameters. This text is shown we use the "?" character.

```
/clispec/$MODE/cmd/options
```

The "options" element specifies under what circumstances the CLI command should execute. It contains (in any order) zero or one "hidden" element, zero or one "hideGroup" element, zero or one "denyRunAccess" element, zero or one "notInterruptible" element, zero or one "pipeFlags" element, zero or one "negPipeFlags" element, zero or one of "submodeCommand" and "topModeCommand", zero or one of "displayWhen" element, and zero or one "paginate" element.

```
/clispec/$MODE/cmd/options/hidden
```

The "hidden" element makes a CLI command invisible even though it can be evaluated if we know about its existence. This comes handy for commands which are used for debugging or are in pre-release state.

```
/clispec/$MODE/cmd/options/hideGroup (xs:string)
```

The "hideGroup" element makes a CLI command invisible and it cannot be used even if a user knows about its existence. The command will become visible when the hide group is 'unhidden' using the unhide command.

This mechanism correspond to the 'tailf:hidden' statement in a YANG module.

`/clispec/operationalMode/cmd/options/denyRunAccess`

The "denyRunAccess" element is used to restrict the possibility to run an operational mode command from configure mode.

Comment: The built-in "run" command is used to execute operational mode commands from configure mode.

`/clispec/$MODE/cmd/options/displayWhen`

The "displayWhen" element can be used to add a displayWhen XPath condition to a command.

Attributes:

expr (xpath expression)

The "expr" attribute is mandatory. It specifies an xpath expression. If the expression evaluates to true then the command is available, otherwise not.

ctx (path)

The "ctx" attribute is optional. If not specified the current editpath/mode-path is used as context node for the xpath evaluation. Note that the xpath expression will automatically evaluate to false if a display when expression is used for a top-level command and no ctx is specified. The path may contain variables defined in the dict.

`/clispec/$MODE/cmd/options/notInterruptible`

The "notInterruptible" element disables <ctrl-c> and the execution of the CLI command can thus not be interrupted.

`/clispec/$MODE/cmd/options/pipeFlags`

The "pipeFlags" element is used to signal that certain pipe commands should be made available if this command is entered.

`/clispec/$MODE/cmd/options/negPipeFlags`

The "negPipeFlags" element is used to signal that certain pipe commands should not be made available if this command is entered, ie it is used to block out specific pipe commands.

`/clispec/$MODE/cmd/options/paginate`

The "paginate" element enables a filter for paging through CLI command output text one screen at a time.

Name

mib_annotations — MIB annotations file format

DESCRIPTION

This manual page describes the syntax and semantics used to write MIB annotations. A MIB annotation file is used to modify the behavior of certain MIB objects without having to edit the original MIB file.

MIB annotations are separate file with a .miba suffix, and is applied to a MIB when a YANG module is generated and when the MIB is compiled. See [ncsc\(1\)](#).

SYNTAX

Each line in a MIB annotation file has the following syntax:

```
<MIB Object Name> <modifier> [= <value>]
```

where `modifier` is one of `max_access`, `display_hint`, `behavior`, `unique`, or `operational`.

Blank lines are ignored, and lines starting with `#` are treated as comments and ignored.

If `modifier` is `max_access`, `value` must be one of `not_accessible` or `read_only`.

If `modifier` is `display_hint`, `value` must be a valid `DISPLAY-HINT` value. The display hint is used to determine if a string object should be treated as text or binary data.

If `modifier` is `behavior`, `value` must be one of `noSuchObject` or `noSuchInstance`. When a YANG module is generated from a MIB, objects with a specified behavior are not converted to YANG. When the SNMP agent responds to SNMP requests for such an object, the corresponding error code is used.

If `modifier` is `unique`, `value` must be a valid YANG "unique" expression, i.e., a space-separated list of column names. This modifier must be given on table entries.

If `modifier` is `operational`, there must not be any value given. A writable object marked as `operational` will be translated into a non-configuration YANG node, marked with a `tailf:writable true` statement, indicating that the object represents writable operational data.

If `modifier` is `sort-priority`, `value` must be a 32 bit integer. The object will be generated with a `tailf:sort-priority` statement. See [tailf_yang_extensions\(5\)](#).

If `modifier` is `ned-modification-dependent`, there must not be any value given. The object will be generated with a `tailf:snmp-ned-modification-dependent` statement. See [tailf_yang_extensions\(5\)](#).

If `modifier` is `ned-set-before-row-modification`, `value` is a valid value for the column. The object will be generated with a `tailf:snmp-ned-set-before-row-modification` statement. See [tailf_yang_extensions\(5\)](#).

If `modifier` is `ned-accessible-column`, `value` refers to a column by name or subid (integer). The object will be generated with a `tailf:snmp-ned-accessible-column` statement. See [tailf_yang_extensions\(5\)](#).

If `modifier` is `ned-delete-before-create`, there must not be any value given. The object will be generated with a `tailf:snmp-ned-delete-before-create` statement. See [tailf_yang_extensions\(5\)](#).

If modifier is `ned-recreate-when-modified`, there must not be any value given. The object will be generated with a `tailf:snmp-ned-recreate-when-modified` statement. See [tailf_yang_extensions\(5\)](#).

EXAMPLE

An example of a MIB annotation file.

```
# the following object does not have value
ifStackLastChange behavior = noSuchInstance

# this deprecated table is not implemented
ifTestTable behavior = noSuchObject
```

SEE ALSO

The NSO User Guide

Name

ncs.conf — NCS daemon configuration file format

DESCRIPTION

Whenever we start (or reload) the NCS daemon it reads its configuration from `./ncs.conf` or `${NCS_DIR}/etc/ncs/ncs.conf` or from the file specified with the `-c` option, as described in [ncs\(1\)](#).

`ncs.conf` is an XML configuration file formally defined by a YANG model, `tailf-ncs-config.yang` as referred to in the SEE ALSO section. This YANG file is included in the distribution. The NCS distribution also includes a commented `ncs.conf.example` file.

A short example: A NCS configuration file which specifies where to find fxs files etc, which facility to use for syslog, that the developer log should be disabled and that the audit log should be enabled. Finally, it also disables clear text NETCONF support:

```
<?xml version="1.0" encoding="UTF-8"?>
<ncs-config xmlns="http://tail-f.com/yang/tailf-ncs-config/1.0">

  <load-path>
    <dir>/etc/ncs</dir>
    <dir>.</dir>
  </load-path>

  <state-dir>/var/ncs/state</state-dir>

  <cdb>
    <db-dir>/var/ncs/cdb</db-dir>
  </cdb>

  <aaa>
    <ssh-server-key-dir>/etc/ncs/ssh</ssh-server-key-dir>
  </aaa>

  <logs>
    <syslog-config>
      <facility>daemon</facility>
    </syslog-config>
    <developer-log>
      <enabled>>false</enabled>
    </developer-log>
    <audit-log>
      <enabled>>true</enabled>
    </audit-log>
  </logs>

  <netconf-north-bound>
    <transport>
      <tcp>
        <enabled>>false</enabled>
      </tcp>
    </transport>
  </netconf-north-bound>

  <webui>
    <transport>
      <tcp>
        <enabled>>false</enabled>
      </tcp>
    </transport>
  </webui>
</ncs-config>
```

```

        <ip>0.0.0.0</ip>
        <port>8008</ip>
    </tcp>
</transport>
</webui>
</ncs-config>

```

Many configuration parameters get their default values as defined in the YANG file. Filename parameters have no default values.

CONFIGURATION PARAMETERS

This section lists all available configuration parameters and their type (within parenthesis) and default values (within square brackets). Parameters are written using a path notation to make it easier to see how they relate to each other.

`/ncs-config`

NCS configuration.

`/ncs-config/db-mode (running) [running]`

This feature is deprecated, NCS supports only running db-mode.

This leaf is not necessary to set, it is kept only for backward compatibility reasons.

`/ncs-config/ncs-ipc-address`

NCS listens by default on 127.0.0.1:4569 for incoming TCP connections from NCS client libraries, such as CDB, MAAPI, the CLI, the external database API, as well as commands from the ncs script (such as 'ncs --reload').

The IP address and port can be changed. If they are changed all clients using MAAPI, CDB et.c. must be re-compiled to handle this. See the deployment user-guide on how to do this.

Note that there are severe security implications involved if NCS is instructed to bind(2) to anything but localhost. Read more about this in the NCS IPC section in the System Managent Topics section of the User Guide. Use the IP 0.0.0.0 if you want NCS to listen(2) on all IPv4 addresses.

`/ncs-config/ncs-ipc-address/ip (ipv4-address | ipv6-address)`
`[127.0.0.1]`

The IP address which NCS listens on for incoming connections from the Java library

`/ncs-config/ncs-ipc-address/port (port-number) [4569]`

The port number which NCS listens on for incoming connections from the Java library

`/ncs-config/ncs-ipc-extra-listen-ip (ipv4-address | ipv6-address)`

This parameter may be given multiple times.

A list of additional IPs to which we wish to bind the NCS IPC listener. This is useful if we don't want to use the wildcard 0.0.0.0 address in order to never expose the NCS IPC to certain interfaces.

`/ncs-config/ncs-ipc-access-check`

NCS can be configured to restrict access for incoming connections to the IPC listener sockets. The access check requires that connecting clients prove possession of a shared secret.

`/ncs-config/ncs-ipc-access-check/enabled (boolean) [false]`

If set to 'true', access check for IPC connections is enabled.

`/ncs-config/ncs-ipc-access-check/filename (string)`

This parameter is mandatory.

filename is the full path to a file containing the shared secret for the IPC access check. The file should be protected via OS file permissions, such that it can only be read by the NCS daemon and client processes that are allowed to connect to the IPC listener sockets.

`/ncs-config/enable-shared-memory-schema (boolean) [true]`

enabled is either true or false. If true, then a C program will be started that loads the schema into shared memory (which then can be accessed by e.g Python)

`/ncs-config/load-path`

`/ncs-config/load-path/dir (string)`

This parameter may be given multiple times.

The load-path element contains any number of dir elements. Each dir element points to a directory path on disk which is searched for compiled and imported YANG files (.fxs files) and compiled clispec files (.ccl files) during daemon startup. NCS also searches the load path for packages at initial startup, or when requested by the `/packages/reload` action.

`/ncs-config/state-dir (string)`

This parameter is mandatory.

This is where NCS writes persistent state data. Currently it is used to store a private copy of all packages found in the load path, in a directory tree rooted at 'packages-in-use.cur' (also referenced by a symlink 'packages-in-use'). It is also used for the state files 'running.invalid', which exists only if the running database status is invalid, which it will be if one of the database implementation fails during the two-phase commit protocol, and 'global.data' which is used to store some data that needs to be retained across reboots.

`/ncs-config/commit-retry-timeout (xs:duration | infinity) [infinity]`

Commit timeout in the NCS backplane. This timeout controls for how long the commit operation in the CLI and the JSON-RPC API will attempt to complete the operation when some other entity is locking the database, e.g. some other commit is in progress or some managed object is locking the database.

`/ncs-config/max-validation-errors (uint32 | unbounded) [1]`

Controls how many validation errors are collected and presented to the user at a time.

`/ncs-config/notifications`

This section defines settings which affect notifications.

NETCONF northbound notification settings

`/ncs-config/notifications/event-streams`

Lists all available notification event streams.

`/ncs-config/notifications/event-streams/stream`

Parameters for a single notification event stream.

`/ncs-config/notifications/event-streams/stream/name (string)`

The name attached to a specific event stream.

`/ncs-config/notifications/event-streams/stream/description (string)`

This parameter is mandatory.

A descriptive text attached to a specific event stream.

`/ncs-config/notifications/event-streams/stream/replay-support (boolean)`

This parameter is mandatory.

Signals if replay support is available for a specific event stream.

/ncs-config/notifications/event-streams/stream/builtin-replay-store

Parameters for the built in replay store for this event stream.

If replay support is enabled NCS automatically stores all notifications on disk ready to be replayed should a NETCONF manager ask for logged notifications. The replay store uses a set of wrapping log files on disk (of a certain number and size) to store the notifications.

The max size of each wrap log file (see below) should not be too large. This to achieve fast replay of notifications in a certain time range. If possible use a larger number of wrap log files instead.

If in doubt use the recommended settings (see below).

/ncs-config/notifications/event-streams/stream/builtin-replay-store/
enabled (boolean) [false]

If set to 'false', the application must implement its own replay support.

/ncs-config/notifications/event-streams/stream/builtin-replay-store/dir
(string)

This parameter is mandatory.

The wrapping log files will be put in this disk location

/ncs-config/notifications/event-streams/stream/builtin-replay-store/
max-size (tailf:size)

This parameter is mandatory.

The max size of each log wrap file. The recommended setting is approximately \$10M.

/ncs-config/notifications/event-streams/stream/builtin-replay-store/
max-files (int64)

This parameter is mandatory.

The max number of log wrap files. The recommended setting is around 50 files.

/ncs-config/opcache

This section defines settings which affect the behavior of the operational data cache.

/ncs-config/opcache/enabled (boolean) [false]

If set to 'true', the cache is enabled.

/ncs-config/opcache/timeout (uint64)

This parameter is mandatory.

The amount of time to keep data in the cache, in seconds.

/ncs-config/hidden-group

Hide groups that can be unhidden must be listed here. There can be zero, one or many hidden-group entries in the configuration.

If a hidden group does not have a hidden-group entry, then it cannot be unhidden using the CLI 'unhide' command. However, it is possible to add a hidden-group entry to the ncs.conf file and then use ncs --reload to make it available in the CLI. This may be useful to enable for example a diagnostics hidden groups that you do not even want accessible using a password.

/ncs-config/hidden-group/name (string)

Name of hidden group. This name should correspond to a hidden group name defined in some YANG module with 'tailf:hidden'.


```
/ncs-config/hide-group/password (tailf:md5-digest-string) []
```

A password can optionally be specified for a hide group. If no password or callback is given then the hide group can be unhidden without giving a password.

If a password is specified then the hide group cannot be enabled unless the password is entered.

To completely disable a hide group, ie make it impossible to unhide it, remove the entire hide-group container for that hide group.

```
/ncs-config/hide-group/callback (string)
```

A callback can optionally be specified for a hide group. If no callback or password is given then the hide group can be unhidden without giving a password.

If a callback is specified then the hide group cannot be enabled unless a password is entered and the successfully verifies the password. The callback receives both the name of the hide group, the name of the user issuing the unhide command, and the password.

Using a callback it is possible to have short lived unhide passwords and per-user unhide passwords.

```
/ncs-config/cdb
```

```
/ncs-config/cdb/db-dir (string)
```

db-dir is the directory on disk which CDB use for its storage and any temporary files being used. It is also the directory where CDB searches for initialization files.

```
/ncs-config/cdb/init-path
```

```
/ncs-config/cdb/init-path/dir (string)
```

This parameter may be given multiple times.

The init-path can contain any number of dir elements. Each dir element points to a directory path which CDB will search for .xml files before looking in db-dir. The directories are searched in the order they are listed.

```
/ncs-config/cdb/client-timeout (xs:duration | infinity) [infinity]
```

Specifies how long CDB should wait for a response to e.g. a subscription notification before considering a client unresponsive. If a client fails to call Cdb.syncSubscriptionSocket() within the timeout period, CDB will syslog this failure and then, considering the client dead, close the socket and proceed with the subscription notifications. If set to infinity, CDB will never timeout waiting for a response from a client.

```
/ncs-config/cdb/subscription-replay
```

```
/ncs-config/cdb/subscription-replay/enabled (boolean) [false]
```

If enabled it is possible to request a replay of the previous subscription notification to a new cdb subscriber.

```
/ncs-config/cdb/replication (async | sync) [sync]
```

When CDB replication is enabled (which it is when high-availability mode is enabled, see /ncs-config/ha) the CDB configuration stores can be replicated either asynchronously or synchronously. With asynchronous replication, a transaction updating the configuration is allowed to complete as soon as the updates have been sent to the connected slaves. With the default synchronous replication, the transaction is suspended until the updates have been completely propagated to the slaves, and the subscribers on the slaves (if any) have acknowledged their subscription notifications

```
/ncs-config/cdb/journal-compaction (automatic | manual) [automatic]
```

Controls the way the CDB configuration store does its journal compaction. Never set to anything but the default 'automatic' unless there is an external mechanism which controls the compaction using the cdb_initiate_journal_compaction() API call.

/ncs-config/cdb/operational

Operational data can either be implemented by external callbacks, or stored in CDB (or a combination of both). The operational datastore is used when data is to be stored in CDB.

/ncs-config/cdb/operational/db-dir (string)

db-dir is the directory on disk which CDB operational uses for its storage and any temporary files being used. If left unset (default) the same directory as db-dir for CDB is used.

/ncs-config/cdb/snapshot

The snapshot datastore is used by the commit queue to calculate the southbound diff towards the devices outside of the transaction lock.

/ncs-config/cdb/snapshot/pre-populate (boolean) [false]

This parameter controls if the snapshot datastore should be pre-populated during upgrade. Switching this on or off implies different trade-offs.

If 'false', NCS is optimized for using normal transaction commits. The snapshot is populated in a lazy manner (when a device is committed through the commit queue for the first time). The drawback is that this commit will suffer performance wise, which is especially true for devices with large configurations. Subsequent commits on the same devices will not have the same penalty.

If 'true', NCS is optimized for systems using the commit queue extensively. This will lead to better performance when committing using the commit queue with no additional penalty for the first time commits. The drawbacks are that upgrade times will increase and an almost doubling of NCS memory consumption.

/ncs-config/encrypted-strings

encrypted-strings defines keys used to encrypt strings adhering to the types tailf:des3-cbc-encrypted-string and tailf:aes-cfb-128-encrypted-string.

/ncs-config/encrypted-strings/DES3CBC

In the DES3CBC case three 64 bits (8 bytes) keys and a random initial vector are used to encrypt the string. The initVector leaf is only used when upgrading from versions before NCS-4.2, but it is kept for backward compatibility reasons.

/ncs-config/encrypted-strings/DES3CBC/key1 (hex8-value-type)

This parameter is mandatory.

/ncs-config/encrypted-strings/DES3CBC/key2 (hex8-value-type)

This parameter is mandatory.

/ncs-config/encrypted-strings/DES3CBC/key3 (hex8-value-type)

This parameter is mandatory.

/ncs-config/encrypted-strings/DES3CBC/initVector (hex8-value-type)

/ncs-config/encrypted-strings/AESCFB128

In the AESCFB128 case one 128 bits (16 bytes) key and a random initial vector are used to encrypt the string. The initVector leaf is only used when upgrading from versions before NCS-4.2, but it is kept for backward compatibility reasons.

/ncs-config/encrypted-strings/AESCFB128/key (hex16-value-type)

This parameter is mandatory.

/ncs-config/encrypted-strings/AESCFB128/initVector (hex16-value-type)

/ncs-config/crypt-hash

crypt-hash specifies how cleartext values should be hashed for leafs of the types ianach:crypt-hash, tailf:sha-256-digest-string, and tailf:sha-512-digest-string.

/ncs-config/crypt-hash/algorithm (md5 | sha-256 | sha-512) [md5]

algorithm can be set to one of the values 'md5', 'sha-256', or 'sha-512', to choose the corresponding hash algorithm for hashing of cleartext input for the ianach:crypt-hash type.

/ncs-config/crypt-hash/rounds (crypt-hash-rounds-type) [5000]

For the 'sha-256' and 'sha-512' algorithms for the ianach:crypt-hash type, and for the tailf:sha-256-digest-string and tailf:sha-512-digest-string types, 'rounds' specifies how many times the hashing loop should be executed. If a value other than the default 5000 is specified, the hashed format will have 'rounds=N\$', where N is the specified value, prepended to the salt. This parameter is ignored for the 'md5' algorithm for ianach:crypt-hash.

/ncs-config/logs

/ncs-config/logs/syslog-config

Shared settings for how to log to syslog. Logs (see below) can be configured to log to file and/or syslog. If a log is configured to log to syslog, the settings under /ncs-config/logs/syslog-config are used.

/ncs-config/logs/syslog-config/version (bsd | 1) [bsd]

version is either 'bsd' (traditional syslog) or '1' (new IETF syslog format: RFC 5424). '1' implies that /ncs-config/logs/syslog-config/udp/enabled must be set to true.

/ncs-config/logs/syslog-config/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32) [daemon]

This facility setting is the default facility. It's also possible to set individual facilities in the different logs below.

/ncs-config/logs/syslog-config/udp

/ncs-config/logs/syslog-config/udp/enabled (boolean) [false]

If set to 'false', messages will be sent to the local syslog daemon.

/ncs-config/logs/syslog-config/udp/host (string | ipv4-address | ipv6-address)

This parameter is mandatory.

host is either a domain name or an IPv4/IPv6 network address. UDP syslog messages are sent to this host.

/ncs-config/logs/syslog-config/udp/port (port-number) [514]

port is a valid port number to be used in combination with /ncs-config/logs/syslog-config/udp/host.

/ncs-config/logs/syslog-config/syslog-servers

This is an alternative way of specifying UDP syslog servers. If we configure the /ncs-config/logs/syslog-config/udp container any configuration in this container is ignored.

/ncs-config/logs/syslog-config/syslog-servers/server

A set of syslog servers that get a copy of all syslog messages.

/ncs-config/logs/syslog-config/syslog-servers/server/host (string | ipv4-address | ipv6-address)

host is either a domain name or an IPv4/IPv6 network address. UDP syslog messages are sent to this host.

```
/ncs-config/logs/syslog-config/syslog-servers/server/port (port-number)
[514]
```

port is the UDP port number where this syslog server is listening.

```
/ncs-config/logs/syslog-config/syslog-servers/server/version (bsd | 1)
[bsd]
```

version is either 'bsd' (traditional syslog) or '1' (new IETF syslog format: RFC 5424).

```
/ncs-config/logs/syslog-config/syslog-servers/server/facility (daemon |
authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6
| local7 | uint32) [daemon]
```

```
/ncs-config/logs/syslog-config/syslog-servers/server/enabled (boolean)
[true]
```

If set to 'false', this syslog server will not get any UDP messages.

```
/ncs-config/logs/ncs-log
```

ncs-log is NCS's daemon log. Check this log for startup problems of the NCS daemon itself. This log is not rotated, i.e. use logrotate(8).

```
/ncs-config/logs/ncs-log/enabled (boolean) [true]
```

If set to true, the log is enabled.

```
/ncs-config/logs/ncs-log/file
```

```
/ncs-config/logs/ncs-log/file/name (string)
```

Name is the full path to the actual log file.

```
/ncs-config/logs/ncs-log/file/enabled (boolean) [false]
```

If set to true, file logging is enabled

```
/ncs-config/logs/ncs-log/syslog
```

```
/ncs-config/logs/ncs-log/syslog/enabled (boolean) [false]
```

If set to true, syslog messages are sent.

```
/ncs-config/logs/ncs-log/syslog/facility (daemon | authpriv | local0 |
local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)
```

This optional value overrides the /ncs-config/logs/syslog-config/facility for this particular log.

```
/ncs-config/logs/developer-log
```

developer-log is a debug log for troubleshooting user-written Java code. Enable and check this log for problems with validation code etc. This log is enabled by default. In all other regards it can be configured as ncs-log. This log is not rotated, i.e. use logrotate(8).

```
/ncs-config/logs/developer-log/enabled (boolean) [true]
```

If set to true, the log is enabled.

```
/ncs-config/logs/developer-log/file
```

```
/ncs-config/logs/developer-log/file/name (string)
```

Name is the full path to the actual log file.

```
/ncs-config/logs/developer-log/file/enabled (boolean) [false]
```

If set to true, file logging is enabled

```
/ncs-config/logs/developer-log/syslog
```

```
/ncs-config/logs/developer-log/syslog/enabled (boolean) [false]
```

If set to true, syslog messages are sent.

```
/ncs-config/logs/developer-log/syslog/facility (daemon | authpriv |  
local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 |  
uint32)
```

This optional value overrides the `/ncs-config/logs/syslog-config/facility` for this particular log.

```
/ncs-config/logs/developer-log-level (error | info | trace) [info]
```

Controls which level of developer messages are printed in the developer log.

```
/ncs-config/logs/audit-log
```

audit-log is an audit log recording successful and failed logins to the NCS backplane. This log is enabled by default. In all other regards it can be configured as `/ncs-config/logs/ncs-log`. This log is not rotated, i.e. use `logrotate(8)`.

```
/ncs-config/logs/audit-log/enabled (boolean) [true]
```

If set to true, the log is enabled.

```
/ncs-config/logs/audit-log/file
```

```
/ncs-config/logs/audit-log/file/name (string)
```

Name is the full path to the actual log file.

```
/ncs-config/logs/audit-log/file/enabled (boolean) [false]
```

If set to true, file logging is enabled

```
/ncs-config/logs/audit-log/syslog
```

```
/ncs-config/logs/audit-log/syslog/enabled (boolean) [false]
```

If set to true, syslog messages are sent.

```
/ncs-config/logs/audit-log/syslog/facility (daemon | authpriv | local0  
| local1 | local2 | local3 | local4 | local5 | local6 | local7 |  
uint32)
```

This optional value overrides the `/ncs-config/logs/syslog-config/facility` for this particular log.

```
/ncs-config/logs/audit-log-commit (boolean) [false]
```

Controls whether the audit log should include messages about the resulting configuration changes for each commit to the running data store.

```
/ncs-config/logs/netconf-log
```

netconf-log is a log for troubleshooting northbound NETCONF operations, such as checking why e.g. a filter operation didn't return the data requested. This log is enabled by default. In all other regards it can be configured as `/ncs-config/logs/ncs-log`. This log is not rotated, i.e. use `logrotate(8)`.

```
/ncs-config/logs/netconf-log/enabled (boolean) [true]
```

If set to true, the log is enabled.

```
/ncs-config/logs/netconf-log/file
```

```
/ncs-config/logs/netconf-log/file/name (string)
```

Name is the full path to the actual log file.

```
/ncs-config/logs/netconf-log/file/enabled (boolean) [false]
```

If set to true, file logging is enabled

```
/ncs-config/logs/netconf-log/syslog
```

```
/ncs-config/logs/netconf-log/syslog/enabled (boolean) [false]
```

If set to true, syslog messages are sent.

```
/ncs-config/logs/netconf-log/syslog/facility (daemon | authpriv |  
local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 |  
uint32)
```

This optional value overrides the `/ncs-config/logs/syslog-config/facility` for this particular log.

```

/ncs-config/logs/snmp-log
/ncs-config/logs/snmp-log/enabled (boolean) [true]
    If set to true, the log is enabled.
/ncs-config/logs/snmp-log/file
/ncs-config/logs/snmp-log/file/name (string)
    Name is the full path to the actual log file.
/ncs-config/logs/snmp-log/file/enabled (boolean) [false]
    If set to true, file logging is enabled
/ncs-config/logs/snmp-log/syslog
/ncs-config/logs/snmp-log/syslog/enabled (boolean) [false]
    If set to true, syslog messages are sent.
/ncs-config/logs/snmp-log/syslog/facility (daemon | authpriv | local0 |
local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)
    This optional value overrides the /ncs-config/logs/syslog-config/facility for this particular log.
/ncs-config/logs/snmp-log-level (error | info) [info]
    Controls which level of SNMP pdus are printed in the SNMP log. The value 'error' means that only
    PDUs with error-status not equal to 'noError' are printed.
/ncs-config/logs/webui-browser-log
    webui-browser-log makes it possible to log Javascript errors/exceptions in a log file on the target
    device instead of just in the browser's error console. This log is not enabled by default and is not
    rotated, i.e. use logrotate(8).

/ncs-config/logs/webui-browser-log/enabled (boolean) [false]
    If set to 'true', the browser log is used.
/ncs-config/logs/webui-browser-log/filename (string)
    This parameter is mandatory.
    The path to the filename where browser log entries should be written

/ncs-config/logs/webui-access-log
    webui-access-log is an access log for the embedded NCS Web server. This file adheres to the
    Common Log Format, as defined by Apache and others. This log is not enabled by default and is not
    rotated, i.e. use logrotate(8).

/ncs-config/logs/webui-access-log/enabled (boolean) [false]
    If set to 'true', the access log is used.
/ncs-config/logs/webui-access-log/traffic-log (boolean) [false]
    Is either true or false. If true, all HTTP(S) traffic towards the embedded Web server is logged in a log
    file named traffic.trace. Beware: Do not use this log in a production setting. This log is not enabled by
    default and is not rotated, i.e. use logrotate(8).
/ncs-config/logs/webui-access-log/dir (string)
    This parameter is mandatory.
    The path to the directory whereas the access log should be written to.

/ncs-config/logs/netconf-trace-log
    netconf-trace-log is a log for understanding and troubleshooting northbound NETCONF protocol
    interactions. When this log is enabled, all NETCONF traffic to and from NCS is stored to a file. By
    default, all XML is pretty-printed. This will slow down the NETCONF server, so be careful when
    enabling this log. This log is not rotated, i.e. use logrotate(8).

```

/ncs-config/logs/netconf-trace-log/enabled (boolean) [false]

If set to 'true', all NETCONF traffic is logged.

/ncs-config/logs/netconf-trace-log/filename (string)

This parameter is mandatory.

The name of the file where the NETCONF traffic trace log is written.

/ncs-config/logs/netconf-trace-log/format (pretty | raw) [pretty]

The value 'pretty' means that the XML data is pretty-printed. The value 'raw' means that it is not.

/ncs-config/logs/xpath-trace-log

xpath-trace-log is a log for understanding and troubleshooting XPath evaluations. When this log is enabled, the execution of all XPath queries evaluated by NCS are logged to a file.

This will slow down NCS, so be careful when enabling this log. This log is not rotated, i.e. use logrotate(8).

/ncs-config/logs/xpath-trace-log/enabled (boolean) [false]

If set to 'true', all XPath execution is logged.

/ncs-config/logs/xpath-trace-log/filename (string)

This parameter is mandatory.

The name of the file where the XPath trace log is written

/ncs-config/logs/error-log

error-log is an error log used for internal logging from the NCS daemon. It is used for troubleshooting the NCS daemon itself, and should normally be disabled. This log is rotated by the NCS daemon (see below).

/ncs-config/logs/error-log/enabled (boolean) [false]

If set to 'true', error logging is performed.

/ncs-config/logs/error-log/filename (string)

This parameter is mandatory.

filename is the full path to the actual log file. This parameter must be set if the error-log is enabled.

/ncs-config/logs/error-log/max-size (tailf:size) [51M]

max-size is the maximum size of an individual log file before it is rotated. Log filenames are reused when five logs have been exhausted.

/ncs-config/logs/error-log/debug

/ncs-config/logs/error-log/debug/enabled (boolean) [false]

/ncs-config/logs/error-log/debug/level (uint16) [2]

/ncs-config/logs/error-log/debug/tag (string)

This parameter may be given multiple times.

/ncs-config/candidate

/ncs-config/candidate/filename (string)

The candidate db-mode has been removed. Hence this leaf no longer affects the NCS configuration.

This leaf and the candidate container is kept only for backward compatibility reasons.

/ncs-config/sort-transactions (boolean) [true]

This parameter controls how NCS lists newly created, not yet committed list entries. If this value is set to 'false', NCS will list all new elements before listing existing data.

If this value is set to 'true', NCS will merge new and existing entries, and provide one sorted view of the data. This behavior works well when CDB is used to store configuration data, but if an external data provider is used, NCS does not know the sort order, and can thus not merge the new entries correctly. If an external data provider is used for configuration data, and the sort order differs from CDB's sort order, this parameter should be set to 'false'.

```
/ncs-config/enable-attributes (boolean) [true]
```

This parameter controls if NCS's attribute feature should be enabled or not. Currently there are two attributes, annotations and tags. These are available in northbound interfaces (e.g. the `annotate` command in the CLI, and `annotation XML attribute` in NETCONF), but in order to be useful they need support from the underlying configuration data provider. CDB supports attributes, but if an external data provider is used for configuration data, and it does not support the attribute callbacks, this parameter should be set to 'false'.

```
/ncs-config/enable-inactive (boolean) [true]
```

This parameter controls if the NCS's inactive feature should be enabled or not. This feature also requires `enableAttributes` to be enabled. When NCS is used to control Juniper routers, this feature is required

```
/ncs-config/session-limits
```

Parameters for limiting concurrent access to NCS.

```
/ncs-config/session-limits/max-sessions (uint32 | unbounded)
[unbounded]
```

Puts a limit to the total number of concurrent sessions to NCS.

```
/ncs-config/session-limits/session-limit
```

Parameters for limiting concurrent access for a specific context to NCS. There can be multiple instances of this container element, each one specifying parameters for a specific context.

```
/ncs-config/session-limits/session-limit/context (string)
```

The context is either one of `cli`, `netconf`, `webui`, `snmp` or it can be any other context string defined through the use of MAAPI. As an example, if we use MAAPI to implement a CORBA interface to NCS, our MAAPI program could send the string `'corba'` as context.

```
/ncs-config/session-limits/session-limit/max-sessions (uint32 |
unbounded)
```

This parameter is mandatory.

Puts a limit to the total number of concurrent sessions to NCS.

```
/ncs-config/session-limits/max-config-sessions (uint32 | unbounded)
[unbounded]
```

Puts a limit to the total number of concurrent configuration sessions to NCS.

```
/ncs-config/session-limits/config-session-limit
```

Parameters for limiting concurrent read-write transactions for a specific context to NCS. There can be multiple instances of this container element, each one specifying parameters for a specific context.

```
/ncs-config/session-limits/config-session-limit/context (string)
```

The context is either one of `cli`, `netconf`, `webui`, `snmp`, or it can be any other context string defined through the use of MAAPI. As an example, if we use MAAPI to implement a CORBA interface to NCS, our MAAPI program could send the string `'corba'` as context.

```
/ncs-config/session-limits/config-session-limit/max-sessions (uint32 |
unbounded)
```

This parameter is mandatory.

Puts a limit to the total number of concurrent configuration sessions to NCS for the corresponding context.

`/ncs-config/aaa`

The login procedure to NCS is fully described in the NCS User Guide.

`/ncs-config/aaa/ssh-login-grace-time (xs:duration) [PT10M]`

NCS servers close ssh connections after this time if the client has not successfully authenticated itself by then. If the value is 0, there is no time limit for client authentication.

This is a global value for all ssh servers in NCS.

Modification of this value will only affect ssh connections that are established after the modification has been done.

`/ncs-config/aaa/ssh-max-auth-tries (uint32 | unbounded) [unbounded]`

NCS servers close ssh connections when the client has made this number of unsuccessful authentication attempts.

This is a global value for all ssh servers in NCS.

Modification of this value will only affect ssh connections that are established after the modification has been done.

`/ncs-config/aaa/ssh-server-key-dir (string)`

ssh-server-key-dir is the directory file path where the keys used by the NCS SSH daemon are found. This parameter must be set if SSH is enabled for NETCONF or the CLI. If SSH is enabled, the server keys used by NCS are on the same format as the server keys used by openssh, i.e. the same format as generated by 'ssh-keygen'

Only DSA- and RSA-type keys can be used with the NCS SSH daemon, as generated by 'ssh-keygen' with the '-t dsa' and '-t rsa' switches, respectively.

The key must be stored with an empty passphrase, and with the name 'ssh_host_dsa_key' if it is a DSA-type key, and with the name 'ssh_host_rsa_key' if it is an RSA-type key.

The SSH server will advertise support for those key types for which there is a key file available and for which the required algorithm is enabled, see the `/ncs-config/ssh/algorithms/server-host-key` leaf.

`/ncs-config/aaa/ssh-publickey-authentication (none | local | system)`
`[system]`

Controls how the NCS SSH daemon locates the user keys for public key authentication.

If set to 'none', public key authentication is disabled.

If set to 'local', and the user exists in `/aaa/authentication/users`, the keys in the user's 'ssh_keydir' directory are used.

If set to 'system', the user is first looked up in `/aaa/authentication/users`, but only if `/ncs-config/aaa/local-authentication/enabled` is set to 'true' - if local-authentication is disabled, or the user does not exist in `/aaa/authentication/users`, but the user does exist in the OS password database, the keys in the user's `$HOME/.ssh` directory are used.

`/ncs-config/aaa/default-group (string)`

If the group of a user cannot be found in the AAA sub-system, a logged in user will end up as a member of the default group (if specified). If a user logs in and the group membership cannot be established, the user will have zero access rights.

`/ncs-config/aaa/auth-order (string)`

The default order for authentication is 'local-authentication pam external-authentication'. It is possible to change this order through this parameter

`/ncs-config/aaa/expiration-warning (ignore | display | prompt) [ignore]`

When PAM or external authentication is used, the authentication mechanism may give a warning that the user's password is about to expire. This parameter controls how the NCS daemon processes that warning message.

If set to 'ignore', the warning is ignored.

If set to 'display', interactive user interfaces will display the warning message at login time.

If set to 'prompt', interactive user interfaces will display the warning message at login time, and require that the user acknowledges the message before proceeding.

`/ncs-config/aaa/audit-user-name (always | known | never) [known]`

Controls the logging of the user name when a failed authentication attempt is logged to the audit log.

If set to "always", the user name is always logged.

If set to "known", the user name is only logged when it is known to be valid (i.e. when attempting local-authentication and the user exists in `/aaa/authentication/users`), otherwise it is logged as "[withheld]".

If set to "never", the user name is always logged as "[withheld]".

`/ncs-config/aaa/pam`

If PAM is to be used for login the NCS daemon typically must run as root.

`/ncs-config/aaa/pam/enabled (boolean) [false]`

When set to 'true', NCS uses PAM for authentication.

`/ncs-config/aaa/pam/service (string) [common-auth]`

The PAM service to be used for the login NETCONF/SSH CLI procedure. This can be any service we have installed in the `/etc/pam.d` directory. Different unices have different services installed under `/etc/pam.d` - choose a service which makes sense or create a new one.

`/ncs-config/aaa/pam/timeout (xs:duration) [PT10S]`

The maximum time that authentication will wait for a reply from PAM. If the timeout is reached, the PAM authentication will fail, but authentication attempts may still be done with other mechanisms as configured for `/ncs-config/aaa/authOrder`. Default is PT10S, i.e. 10 seconds.

`/ncs-config/aaa/external-authentication`

`/ncs-config/aaa/external-authentication/enabled (boolean) [false]`

When set to 'true', external authentication is used.

`/ncs-config/aaa/external-authentication/executable (string)`

If we enable external authentication, an executable on the local host can be launched to authenticate a user. The executable will receive the username and the cleartext password on its standard input. The format is `'[${USER}];[${PASS}];\n'`. For example if user is 'bob' and password is 'secret', the executable will receive the line `'[bob;secret;]'` followed by a newline on its standard input. The program must parse this line.

The task of the external program, which for example could be a RADIUS client is to authenticate the user and also provide the user to groups mapping. So if 'bob' is member

of the 'oper' and the 'lamers' group, the program should echo 'accept oper lamers' on its standard output. If the user fails to authenticate, the program should echo 'reject \${reason}' on its standard output.

```
/ncs-config/aaa/external-authentication/use-base64 (boolean) [false]
```

When set to 'true', \${USER} and \${PASS} in the data passed to the executable will be base64-encoded, allowing e.g. for the password to contain ';' characters. For example if user is 'bob' and password is 'secret', the executable will receive the string '[Ym9i;c2Vjc2V0;]' followed by a newline.

```
/ncs-config/aaa/external-authentication/include-extra (boolean) [false]
```

When set to 'true', additional information items will be provided to the executable: source IP address and port, context, and protocol. I.e. the complete format will be '[\${USER};\${PASS};\${IP};\${PORT};\${CONTEXT};\${PROTO};]\n'. Example: '[bob;secret;192.168.1.1;12345;cli;ssh;]\n'.

```
/ncs-config/aaa/local-authentication
```

```
/ncs-config/aaa/local-authentication/enabled (boolean) [true]
```

When set to true, NCS uses local authentication. That means that the user data kept in the aaa namespace is used to authenticate users. When set to false some other authentication mechanism such as PAM or external authentication must be used.

```
/ncs-config/aaa/authentication-callback
```

```
/ncs-config/aaa/authentication-callback/enabled (boolean) [false]
```

When set to true, NCS will invoke an application callback when authentication has succeeded or failed. The callback may reject an otherwise successful authentication. If the callback has not been registered, all authentication attempts will fail. See Javadoc for DpAuthCallback for the callback details.

```
/ncs-config/aaa/authorization
```

```
/ncs-config/aaa/authorization/enabled (boolean) [true]
```

When set to false, all authorization checks are turned off, similar to the -noaaa flag in ncs_cli.

```
/ncs-config/aaa/authorization/callback
```

```
/ncs-config/aaa/authorization/callback/enabled (boolean) [false]
```

When set to true, NCS will invoke application callbacks for authorization. If the callbacks have not been registered, all authorization checks will be rejected. See Javadoc for DpAuthorizationCallback for the callback details.

```
/ncs-config/aaa/namespace (string) [http://tail-f.com/ns/aaa/1.1]
```

If we want to move the AAA data into another userdefine namespace, we indicate that here.

```
/ncs-config/aaa/prefix (string) [/]
```

If we want to move the AAA data into another userdefined namespace, we indicate the prefix path in that namespace where the NCS AAA namespace has been mounted.

```
/ncs-config/rollback
```

Settings controlling if and where rollback files are created. A rollback file contains a copy of the system configuration. The current running configuration is always stored in rollback0, the previous version in rollback1 etc. The oldest saved configuration has the highest suffix.

```
/ncs-config/rollback/enabled (boolean) [false]
```

When set to true a rollback file will be created whenever the running configuration is modified.

`/ncs-config/rollback/directory (string)`

This parameter is mandatory.

Location where rollback files will be created.

`/ncs-config/rollback/history-size (uint32) [35]`

Number of old configurations to save.

`/ncs-config/rollback/type (delta) [delta]`

This parameter is deprecated, NCS supports only type 'delta'. It is not necessary to set a value for this parameter, it is kept only for backward compatibility reasons.

Type 'delta' means that only the changes are stored in the rollback file. Rollback file 0 will contain the changes from the last configuration commit. This is space and time efficient for large configurations.

`/ncs-config/rollback/rollback-numbering (rolling | fixed) [fixed]`

rollbackNumbering is either 'fixed' or 'rolling'. If set to 'rolling' then rollback file '0' will always contain the last commit. When using 'fixed' each rollback will get a unique increasing number.

`/ncs-config/ssh`

This section defines settings which affect the behavior of the SSH server built into NCS.

`/ncs-config/ssh/idle-connection-timeout (xs:duration) [PT10M]`

The maximum time that an authenticated connection to the SSH server is allowed to exist without open channels. If the timeout is reached, the SSH server closes the connection. Default is PT10M, i.e. 10 minutes. If the value is 0, there is no timeout.

`/ncs-config/ssh/algorithms`

This section defines custom lists of algorithms to be usable with the built-in SSH implementation.

For each type of algorithm, an empty value means that all supported algorithms should be usable, and a non-empty value (a comma-separated list of algorithm names) means that the intersection of the supported algorithms and the configured algorithms should be usable.

`/ncs-config/ssh/algorithms/server-host-key (string) []`

The supported serverHostKey algorithms (if implemented in libcrypto) are "ssh-dss" and "ssh-rsa", but for any SSH server, it is limited to those algorithms for which there is a host key installed in the directory given by `/ncs-config/aaa/ssh-server-key-dir`.

To limit the usable serverHostKey algorithms to "ssh-dss", set this value to "ssh-dss" or avoid installing a key of any other type than ssh-dss in the `sshServerKeyDir`.

`/ncs-config/ssh/algorithms/kex (string) []`

The supported key exchange algorithms (as long as their hash functions are implemented in libcrypto) are "diffie-hellman-group-exchange-sha256", "diffie-hellman-group-exchange-sha1", "diffie-hellman-group14-sha1" and "diffie-hellman-group1-sha1".

To limit the usable key exchange algorithms to "diffie-hellman-group14-sha1" and "diffie-hellman-group-exchange-sha256" (in that order) set this value to "diffie-hellman-group14-sha1, diffie-hellman-group-exchange-sha256".

`/ncs-config/ssh/algorithms/dh-group`

Range of allowed group size, the SSH server responds to the client during a "diffie-hellman-group-exchange". The range will be the intersection of what the client requests, if there is none the key exchange will be aborted.

`/ncs-config/ssh/algorithms/dh-group/min-size (dh-group-size-type)`
`[2048]`

Minimal size of p in bits.

```

/ncs-config/ssh/algorithms/dh-group/max-size (dh-group-size-type)
[4096]
    Maximal size of p in bits.
/ncs-config/ssh/algorithms/mac (string) []
    The supported mac algorithms (if implemented in libcrypto) are "hmac-md5", "hmac-sha1", "hmac-
    sha2-256", "hmac-sha2-512", "hmac-sha1-96" and "hmac-md5-96".
/ncs-config/ssh/algorithms/encryption (string) []
    The supported encryption algorithms (if implemented in libcrypto) are "aes128-ctr", "aes192-ctr",
    "aes256-ctr", "aes128-cbc", "aes256-cbc" and "3des-cbc".
/ncs-config/ssh/client-alive-interval (xs:duration | infinity)
[infinity]
    If no data has been received from a connected client for this long, a request that requires a response
    from the client, will be sent over the SSH transport.
/ncs-config/ssh/client-alive-count-max (uint32) [3]
    If no data has been received from the client, after this many consecutive client-alive-interval has
    passed, the connection will be dropped.
/ncs-config/cli
    CLI parameters.

/ncs-config/cli/enabled (boolean) [true]
    When set to true, the CLI server is started.
/ncs-config/cli/allow-implicit-wildcard (boolean) [true]
    When set to true, users do not need to explicitly type * in the place of keys in lists, in order to see all
    list instances. When set to false, users have to explicitly type * to see all list instances.
/ncs-config/cli/completion-show-max (cli-max) [100]
    Maximum number of possible alternatives to present when doing completion.
/ncs-config/cli/style (j | c)
    Style is either 'j', or 'c'. If set to 'j' then the CLI will be presented as a Juniper style CLI. If 'c' then the
    CLI will appear as Cisco XR style
/ncs-config/cli/ssh
/ncs-config/cli/ssh/enabled (boolean) [true]
    enabled is either 'true' or 'false'. If 'true' the NCS CLI will use the built in SSH server.
/ncs-config/cli/ssh/ip (ipv4-address | ipv6-address) [0.0.0.0]
    ip is an IP address which the NCS CLI should listen on for SSH connections. 0.0.0.0 means that it
    listens on the port (/ncs-config/cli/ssh/port) for all IPv4 addresses on the machine.
/ncs-config/cli/ssh/port (port-number) [2024]
    The port number for CLI SSH
/ncs-config/cli/ssh/banner (string) []
    banner is a string that will be presented to the client before authenticating when logging in to the CLI
    via the built-in SSH server.
/ncs-config/cli/ssh/banner-file (string) []
    banner-file is the name of a file whose contents will be presented (after any string given by the banner
    directive) to the client before authenticating when logging in to the CLI via the built-in SSH server.
/ncs-config/cli/ssh/extra-listen
    A list of additional IP address and port pairs which the NCS CLI should also listen on for SSH
    connections.

```

```
/ncs-config/cli/ssh/extra-listen/ip (ipv4-address | ipv6-address)
```

```
/ncs-config/cli/ssh/extra-listen/port (port-number)
```

```
/ncs-config/cli/top-level-cmds-in-sub-mode (boolean) [false]
```

topLevelCmdsInSubMode is either 'true' or 'false'. If set to 'true' all top level commands in I and C-style CLI are available in sub modes.

```
/ncs-config/cli/completion-meta-info (false | alt1 | alt2) [false]
```

completionMetaInfo is either 'false', 'alt1' or 'alt2'. If set to 'alt1' then the alternatives shown for possible completions will be prefixed as follows:

containers with > lists with + leaf-lists +

for example

Possible completions: ... > applications + apply-groups ... + dns-servers ...

If set to 'alt2', then possible completions will be prefixed as follows

containers with > lists with children with +> lists without children +

for example

Possible completions: ... > applications +>apply-groups ... + dns-servers ...

```
/ncs-config/cli/allow-abbrev-keys (boolean) [false]
```

allowAbbrevKeys is either 'true' or 'false'. If 'false' then key elements are not allowed to be abbreviated in the CLI. This is relevant in the J-style CLI when using the commands 'delete' and 'edit'. In the C/I-style CLIs when using the commands 'no', 'show configuration' and for commands to enter submodes.

```
/ncs-config/cli/j-align-leaf-values (boolean) [true]
```

j-align-leaf-values is either 'true' or 'false'. If 'true' then the leaf values of all siblings in a container or list will be aligned.

```
/ncs-config/cli/enter-submode-on-leaf (boolean) [true]
```

enterSubmodeOnLeaf is either 'true' or 'false'. If set to 'true' (the default) then setting a leaf in a submode from a parent mode results in entering the submode after the command has completed. If set to 'false' then an explicit command for entering the submode is needed. For example, if running the command

```
interface FastEthernet 1/1/1 mtu 1400
```

from the top level in config mode. If enterSubmodeOnLeaf is true the CLI will end up in the 'interface FastEthernet 1/1/1' submode after the command execution. If set to 'false' then the CLI will remain at the top level. To enter the submode when set to 'false' the command

```
interface FastEthernet 1/1/1
```

is needed. Applied to the C-style CLI.

```
/ncs-config/cli/table-look-ahead (int64) [50]
```

The tableLookAhead element tells confd how many rows to pre-fetch when displaying a table. The prefetched rows are used for calculating the required column widths for the table. If set to a small number it is recommended to explicitly configure the column widths in the clispec file.

```
/ncs-config/cli/more-buffer-lines (uint32 | unbounded) [unbounded]
```

moreBufferLines is used to limit the buffering done by the more process. It can be 'unbounded' or a positive integer describing the maximum number of lines to buffer.

/ncs-config/cli/show-all-ns (boolean) [false]

If showAllNs is true then all elem names will be prefixed with the namespace prefix in the CLI. This is visible when setting values and when showing the configuration

/ncs-config/cli/suppress-fast-show (boolean) [false]

suppressFastShow is either 'true' or 'false'. If 'true' then the fast show optimization will be suppressed in the C-style CLI. The fast show optimization is somewhat experimental and may break certain operations.

/ncs-config/cli/use-expose-ns-prefix (boolean) [true]

If 'true' then all nodes annotated with the tailf:cli-expose-ns-prefix will result in the namespace prefix being shown/required. If set to 'false' then the tailf:cli-expose-ns-prefix annotation will be ignored. The container /devices/device/config has this annotation.

/ncs-config/cli/show-defaults (boolean) [false]

show-defaults is either 'true' or 'false'. If 'true' then default values will be shown when displaying the configuration. The default value is shown inside a comment on the same line as the value. Showing default values can also be enabled in the CLI per session using the operational mode command 'set show defaults true'.

/ncs-config/cli/default-prefix (string) []

default-prefix is a string that is placed in front of the default value when a configuration is shown with default values as comments.

/ncs-config/cli/commit-retry-timeout (xs:duration | infinity) [PT0S]

Commit timeout in the CLI. This timeout controls for how long the commit operation will attempt to complete the operation when some other entity is locking the database, e.g. some other commit is in progress or some managed object is locking the database.

There is a similar configuration parameter, /ncs-config/commit-retry-timeout, which sets a timeout for NCS transactions in the JSON-RPC API.

/ncs-config/cli/timezone (utc | local) [local]

Time in the CLI can be either local, as configured on the host, or UTC.

/ncs-config/cli/with-defaults (boolean) [false]

withDefaults is either 'true' or 'false'. If 'false' then leaf nodes that have their default values will not be shown when the user displays the configuration, unless the user gives the 'details' option to the 'show' command.

This is useful when there are many settings which are seldom used. When set to 'false' only the values actually modified by the user will be shown.

/ncs-config/cli/banner (string) []

Banner shown to the user when the CLI is started. Default is empty.

/ncs-config/cli/banner-file (string) []

File whose contents are shown to the user (after any string set by the 'banner' directive) when the CLI is started. Default is empty.

/ncs-config/cli/prompt1 (string) [\u@\h\M>]

Prompt used in operational mode. The string may contain a number of backslash-escaped special characters which are decoded as follows: \d the date in 'YYYY-MM-DD' format (e.g., '2006-01-18') \h the hostname up to the first '.' (or delimiter as defined by promptHostnameDelimiter) \H the hostname the current time in 24-hour HH:MM:SS format \T the current time in 12-hour HH:MM:SS format \@ the current time in 12-hour am/pm format \A the current time in 24-hour HH:MM format \u the username of the current user \m the mode name (only used in XR style) \M the mode name inside parenthesis if in a mode

```

/ncs-config/cli/prompt2 (string) [\u@\h\M% ]
    Prompt used in configuration mode. The string may contain a number of backslash-escaped special
    characters which are decoded as described for prompt1.
/ncs-config/cli/c-prompt1 (string) [\u@\h\M> ]
    Prompt used in operational mode in the Cisco XR style CLI. The string may contain a number of
    backslash-escaped special characters which are decoded as described for prompt1.
/ncs-config/cli/c-prompt2 (string) [\u@\h\M% ]
    Prompt used in configuration mode in the Cisco XR style CLI. The string may contain a number of
    backslash-escaped special characters which are decoded as described for prompt1.
/ncs-config/cli/prompt-hostname-delimiter (string) [. ]
    When the \h token is used in a prompt the first part of the hostname up until the first occurrence of the
    promptHostnameDelimiter is used.
/ncs-config/cli/show-log-directory (string) [/var/log]
    Location where the 'show log' command looks for log files.
/ncs-config/cli/idle-timeout (xs:duration) [PT30M]
    Maximum idle time before terminating a CLI session. Default is PT30M, ie 30 minutes.
/ncs-config/cli/prompt-sessions-cli (boolean) [false]
    promptSessionsCLI is either 'true' or 'false'. If set to 'true' then only the current CLI sessions will
    be displayed when the user tries to start a new CLI session and the maximum number of sessions
    has been reached. Note that MAAPI sessions with their context set to 'cli' would be regarded as CLI
    sessions and would be listed as such.
/ncs-config/cli/suppress-ned-errors (boolean) [false]
    Suppress errors from NED devices. Make log-communication between ncs and its devices more silent.
    Be cautious with this option since errors that might be interesting can get suppressed as well.
/ncs-config/cli/disable-idle-timeout-on-cmd (boolean) [true]
    disable-idle-timeout-on-cmd is either 'true' or 'false'. If set to 'false' then the idle timeout will trigger
    even when a command is running in the CLI. If set to 'true' the idle timeout will only trigger if the user
    is idling at the CLI prompt.
/ncs-config/cli/command-timeout (xs:duration | infinity) [infinity]
    Global command timeout. Terminate command unless the command has completed within the
    timeout. It is generally a bad idea to use this feature since it may have undesirable effects in a loaded
    system where normal commands take longer to complete than usual.
    This timeout can be overridden by a command specific timeout specified in the ncs.cli file.

/ncs-config/cli/space-completion
/ncs-config/cli/space-completion/enabled (boolean)
/ncs-config/cli/ignore-leading-whitespace (boolean)
    If 'false' then the CLI will show completion help when the user enters TAB or SPACE as the first
    characters on a row. If set to 'true' then leading SPACE and TAB are ignored. The user can enter '?' to
    get a list of possible alternatives. Setting the value to 'true' makes it easier to paste scripts into the CLI.
/ncs-config/cli/auto-wizard
    Default value for autowizard in the CLI. The user can always enable or disable the auto wizard in each
    session, this controls the initial session value.

/ncs-config/cli/auto-wizard/enabled (boolean) [true]
    enabled is either 'true' or 'false'. If 'true' the CLI will prompt the user for required attributes when a
    new identifier is created.

```



```

/ncs-config/cli/restricted-file-access (boolean) [false]
    restricted-file-access is either 'true' or 'false'. If 'true' then a CLI user will not be able to access files and
    directories outside the home directory tree.
/ncs-config/cli/restricted-file-regexp (string) []
    restricted-file-regexp is either an empty string or an regular expression (AWK style). If not empty then
    all files and directories created or accessed must match the regular expression. This can be used to
    ensure that certain symbols does not occur in created files.
/ncs-config/cli/history-save (boolean) [true]
    If set to 'true' then the CLI history will be saved between CLI sessions. The history is stored in the
    state directory.
/ncs-config/cli/history-remove-duplicates (boolean) [false]
    If set to 'true' then repeated commands in the CLI will only be stored once in the history. Each
    invocation of the command will only update the date of the last entry. If set to 'false' duplicates will be
    stored in the history.
/ncs-config/cli/history-max-size (int64) [1000]
    Sets maximum configurable history size.
/ncs-config/cli/message-max-size (int64) [10000]
    Maximum size of user message.
/ncs-config/cli/show-commit-progress (boolean) [true]
    show-commit-progress can be either 'true' or 'false'. If set to 'true' then the commit operation in the CLI
    will provide some progress information.
/ncs-config/cli/commit-message (boolean) [true]
    CLI prints out a message when a commit is executed
/ncs-config/cli/use-double-dot-ranges (boolean) [true]
    useDoubleDotRanges is either 'true' or 'false'. If 'true' then range expressions are types as 1..3, if set to
    'false' then ranges are given as 1-3.
/ncs-config/cli/allow-range-expression-all-types (boolean) [true]
    allowRangeExpressionAllTypes is either 'true' or 'false'. If 'true' then range expressions are allowed for
    all key values regardless of type.
/ncs-config/cli/suppress-range-keyword (boolean) [false]
    suppressRangeKeyword is either 'true' or 'false'. If 'true' then 'range' keyword is not allowed in C- and
    I-style for range expressions.
/ncs-config/cli/commit-message-format (string) [ System message at
$(time)... Commit performed by $(user) via $(proto) using $(ctx). ]
    The format of the CLI commit messages
/ncs-config/cli/suppress-commit-message-context (string)
    This parameter may be given multiple times.
    A list of contexts for which no commit message shall be displayed. A good value is [ system ] which
    will make all system generated commits to go unnoticed in the CLI. A context is either the name of an
    agent i.e cli, webui, netconf, snmp or any free form text string if the transaction is initiated from Maapi

/ncs-config/cli/show-subsystem-messages (boolean) [true]
    show-subsystem-messages is either 'true' or 'false'. If 'true' the CLI will display a system message
    whenever a connected daemon is started or stopped.
/ncs-config/cli/show-editors (boolean) [true]
    show-editors is either 'true' or 'false'. If set to true then a list of current editors will be displayed when a
    user enters configure mode.

```

/ncs-config/cli/rollback-aaa (boolean) [false]

If set to true then AAA rules will be applied when a rollback file is loaded. This means that rollback may not be possible if some other user have made changes that the current user does not have access privileges to.

/ncs-config/cli/rollback-numbering (rolling | fixed) [fixed]

rollbackNumbering is either 'fixed' or 'rolling'. If set to 'rolling' then rollback file '0' will always contain the last commit. When using 'fixed' each rollback will get a unique increasing number.

/ncs-config/cli/show-service-meta-data (boolean) [false]

If set to true, then backpointers and refcounts are displayed by default when showing the configuration. If set to false, they are not. The default can be overridden by the pipe flags 'display service-meta' and 'hide service-meta'.

/ncs-config/cli/escape-backslash (boolean) [false]

escapeBackslash is either 'true' or 'false'. If 'true' then backslash is escaped in the CLI.

/ncs-config/cli/show-ned-error-as-info (boolean) [true]

showNedErrorAsInfo is either 'true' or 'false'. If 'true' then error from NED devices is shown as info in the trace log when CLI fails to parse unmodeled config from NED devices.

/ncs-config/rest

This section defines settings which decide how the embedded NCS Web server should behave, with respect to TCP and SSL etc.

/ncs-config/rest/enabled (boolean) [false]

enabled is either 'true' or 'false'. If 'true', the Web server is started.

/ncs-config/rest/custom-headers

/ncs-config/rest/custom-headers/header

/ncs-config/rest/custom-headers/header/name (string)

/ncs-config/rest/custom-headers/header/value (string)

This parameter is mandatory.

/ncs-config/restconf

This section defines settings for the RESTCONF api.

/ncs-config/restconf/enabled (boolean) [false]

enabled is either 'true' or 'false'. If 'true', the RESTCONF api is enabled on the Web server used by the Web UI. Note that the Web UI must be enabled as well.

/ncs-config/restconf/root-resource (string) [restconf]

The RESTCONF root resource path.

/ncs-config/webui

This section defines settings which decide how the embedded NCS Web server should behave, with respect to TCP and SSL etc.

/ncs-config/webui/custom-headers

The custom-headers element contains any number of header elements, with a valid header-field as defined in RFC7230.

The headers will be part of HTTP responses on '/login.html', '/index.html' and '/jsonrpc'.

/ncs-config/webui/custom-headers/header

/ncs-config/webui/custom-headers/header/name (string)

/ncs-config/webui/custom-headers/header/value (string)

This parameter is mandatory.

/ncs-config/webui/enabled (boolean) [false]

enabled is either 'true' or 'false'. If 'true', the Web server is started.

/ncs-config/webui/server-name (string) [localhost]

The hostname the Web server serves.

/ncs-config/webui/match-host-name (boolean) [false]

This setting specifies if the Web server only should serve URLs adhering to the serverName defined above. By default the server-name is 'localhost' and match-host-name is 'false', i.e. any server name can be given in the URL. If you want the server to only accept URLs adhering to the server-name, enable this setting.

/ncs-config/webui/cache-refresh-secs (uint64) [0]

The NCS Web server uses a RAM cache for static content. An entry sits in the cache for a number of seconds before it is reread from disk (on access). The default is 0.

/ncs-config/webui/max-ref-entries (uint64) [100]

Leafref and keyref entries are represented as drop-down menus in the automatically generated Web UI. By default no more than 100 entries are fetched. This element makes this number configurable.

/ncs-config/webui/docroot (string)

The location of the document root on disk. If this configurable is omitted the docroot points to the next generation docroot in the NCS distro instead.

/ncs-config/webui/webui-index-url (string) [/index.html]

Where to redirect after successful login, which by default is '/index.html'.

/ncs-config/webui/hatchery-url (string) [/webui]

Url where the 'hatchery' webui is mapped if webui is enabled. The default is '/webui'.

/ncs-config/webui/login-dir (string)

The login-dir element points out an alternative login directory which contains your HTML code etc used to login to the Web UI. This directory will be mapped https://<ip-address>/login. If this element is not specified the default login/ directory in the docroot will be used instead.

/ncs-config/webui/X-Frame-Options (DENY | SAMEORIGIN | ALLOW-FROM)
[DENY]

By default the X-Frame-Options header is set to DENY for the /login.html and /index.html pages.

With this header it can be set to SAMEORIGIN or ALLOW-FROM instead.

/ncs-config/webui/disable-auth

/ncs-config/webui/disable-auth/dir (string)

This parameter may be given multiple times.

The disable-auth element contains any number of dir elements. Each dir element points to a directory path in the docroot which should not be restricted by the AAA engine. If no dir elements are specified the following directories and files will not be restricted by the AAA engine: '/login' and '/login.html'.

/ncs-config/webui/allow-symlinks (boolean) [true]

Allow symlinks in the docroot directory.

/ncs-config/webui/transport

Settings deciding which transport services the Web server should listen on, e.g. TCP and SSL.

/ncs-config/webui/transport/tcp

Settings deciding how the Web server TCP transport service should behave.

```

/ncs-config/webui/transport/tcp/enabled (boolean) [true]
    enabled is either 'true' or 'false'. If 'true', the Web server uses clear text TCP as a transport service.
/ncs-config/webui/transport/tcp/redirect (string)
    If given the user will be redirected to the specified URL. Two macros can be specified, i.e. @HOST@
    and @PORT@. For example https://@HOST@:443 or https://192.12.4.3:@PORT@
/ncs-config/webui/transport/tcp/ip (ipv4-address | ipv6-address)
[0.0.0.0]
    The IP address which the Web server should listen on. 0.0.0.0 means that it listens on the port (/ncs-
    config/webui/transport/tcp/port) for all IPv4 addresses on the machine.
/ncs-config/webui/transport/tcp/port (port-number) [8008]
    port is a valid port number to be used in combination with the address in /ncs-config/webui/transport/
    tcp/ip.
/ncs-config/webui/transport/tcp/extra-listen
    A list of additional IP address and port pairs which the Web server should also listen on.
/ncs-config/webui/transport/tcp/extra-listen/ip (ipv4-address | ipv6-
address)
/ncs-config/webui/transport/tcp/extra-listen/port (port-number)
/ncs-config/webui/transport/ssl
    Settings deciding how the Web server SSL (Secure Sockets Layer) transport service should behave.
    SSL is widely deployed on the Internet and virtually all bank transactions as well as all on-line
    shopping today is done with SSL encryption. There are many good sources on describing SSL in
    detail, e.g. http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/ which describes how to manage
    certificates and keys.

/ncs-config/webui/transport/ssl/enabled (boolean) [false]
    enabled is either 'true' or 'false'. If 'true', the Web server uses SSL as a transport service.
/ncs-config/webui/transport/ssl/redirect (string)
    If given the user will be redirected to the specified URL. Two macros can be specified, i.e. @HOST@
    and @PORT@. For example http://@HOST@:80 or http://192.12.4.3:@PORT@
/ncs-config/webui/transport/ssl/ip (ipv4-address | ipv6-address)
[0.0.0.0]
    The IP address which the Web server should listen on for incoming ssl connections. 0.0.0.0 means that
    it listens on the port (/ncs-config/webui/transport/ssl/port) for all IPv4 addresses on the machine.
/ncs-config/webui/transport/ssl/port (port-number) [8888]
    port is a valid port number to be used in combination with /ncs-config/webui/transport/ssl/ip.
/ncs-config/webui/transport/ssl/extra-listen
    A list of additional IP address and port pairs which the Web server should also listen on for incoming
    ssl connections.
/ncs-config/webui/transport/ssl/extra-listen/ip (ipv4-address | ipv6-
address)
/ncs-config/webui/transport/ssl/extra-listen/port (port-number)
/ncs-config/webui/transport/ssl/key-file (string)
    Specifies which file that contains the private key for the certificate. Read more about certificates in /
    ncs-config/webui/transport/ssl/cert-file. If this configurable is omitted the keyFile points to a built-
    in self signed certificate/key in the NCS distro instead. Note: Only use this certificate/key for test
    purposes.

```

/ncs-config/webui/transport/ssl/cert-file (string)

Specifies which file that contains the server certificate. The certificate is either a self-signed test certificate or a genuine and validated certificate bought from a CA (Certificate Authority). If this configurable is omitted the keyFile points to a built-in self signed certificate/key in the NCS distro instead. Note: Only use this certificate/key for test purposes.

The NCS distro comes with a server certificate which can be used for testing purposes
(\${NCS_DIR}/var/ncs/webui/cert/host.{cert,key}). This server certificate has been generated using a local CA certificate:

```
$ openssl OpenSSL> genrsa -out ca.key 4096 OpenSSL> req -new -x509 -days 3650 -key ca.key -  
out ca.cert OpenSSL> genrsa -out host.key 4096 OpenSSL> req -new -key host.key -out host.csr  
OpenSSL> x509 -req -days 365 -in host.csr -CA ca.cert \ -CAkey ca.key -set_serial 01 -out host.cert
```

/ncs-config/webui/transport/ssl/ca-cert-file (string)

Specifies which file that contains the trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable CA certificates passed to the client when a certificate is requested.

The NCS distro comes with a CA certificate which can be used for testing purposes
(\${NCS_DIR}/var/ncs/webui/ca_cert/ca.cert).
This CA certificate has been generated as shown above.

/ncs-config/webui/transport/ssl/verify (1 | 2 | 3) [1]

Specifies the level of verification the server does on client certificates. 1 means nothing, 2 means the server will ask the client for a certificate but not fail if the client does not supply a client certificate, 3 means that the server requires the client to supply a client certificate.

If ca-cert-file has been set to the ca.cert file generated above you can verify that it works correctly using, for example:

```
$ openssl s_client -connect 127.0.0.1:8888 \ -cert client.cert -key client.key
```

For this to work client.cert must have been generated using the ca.cert from above:

```
$ openssl OpenSSL> genrsa -out client.key 4096 OpenSSL> req -new -key client.key -out client.csr  
OpenSSL> x509 -req -days 3650 -in client.csr -CA ca.cert \ -CAkey ca.key -set_serial 01 -out  
client.cert
```

/ncs-config/webui/transport/ssl/depth (uint64) [1]

Specifies the depth of certificate chains the server is prepared to follow when verifying client certificates.

/ncs-config/webui/transport/ssl/ciphers (string) [DEFAULT]

Specifies the cipher suites to be used by the server as a colon-separated list from the set ECDHE-ECDSA-AES256-SHA384, ECDHE-RSA-AES256-SHA384, ECDH-ECDSA-AES256-SHA384, ECDH-RSA-AES256-SHA384, DHE-RSA-AES256-SHA256, DHE-DSS-AES256-SHA256, AES256-SHA256, ECDHE-ECDSA-AES128-SHA256, ECDHE-RSA-AES128-SHA256, ECDH-ECDSA-AES128-SHA256, ECDH-RSA-AES128-SHA256, DHE-RSA-AES128-SHA256, DHE-DSS-AES128-SHA256, AES128-SHA256, ECDHE-ECDSA-AES256-SHA, ECDHE-RSA-AES256-SHA, DHE-RSA-AES256-SHA, DHE-DSS-AES256-SHA, ECDH-ECDSA-AES256-SHA, ECDH-RSA-AES256-SHA, AES256-SHA, ECDHE-ECDSA-DES-CBC3-SHA, ECDHE-RSA-DES-CBC3-SHA, EDH-RSA-DES-CBC3-SHA, EDH-DSS-DES-CBC3-SHA, ECDH-ECDSA-DES-CBC3-SHA, ECDH-RSA-DES-CBC3-SHA, DES-CBC3-SHA, ECDHE-ECDSA-AES128-SHA, ECDHE-RSA-

AES128-SHA, DHE-RSA-AES128-SHA, DHE-DSS-AES128-SHA, ECDH-ECDSA-AES128-SHA, ECDH-RSA-AES128-SHA, AES128-SHA, EDH-RSA-DES-CBC-SHA, and DES-CBC-SHA, or the word "DEFAULT" (use the listed set except the suites using DES or MD5 algorithms). See the OpenSSL manual page `ciphers(1)` for the definition of the cipher suites. NOTE: The general cipher list syntax described in `ciphers(1)` is not supported.

`/ncs-config/webui/transport/ssl/protocols (string) [DEFAULT]`

Specifies the SSL/TLS protocol versions to be used by the server as a whitespace-separated list from the set `ssl3 tlsv1 tlsv1.1 tlsv1.2`, or the word "DEFAULT" (use all supported protocol versions except `ssl3`).

`/ncs-config/webui/cgi`

CGI-script support

`/ncs-config/webui/cgi/enabled (boolean) [false]`

`enabled` is either 'true' or 'false'. If 'true', CGI-script support is enabled.

`/ncs-config/webui/cgi/dir (string) [cgi-bin]`

The directory path to the location of the CGI-scripts.

`/ncs-config/webui/cgi/request-filter (string)`

Specifies that characters not specified in the given regexp should be filtered out silently.

`/ncs-config/webui/cgi/max-request-length (uint16)`

Specifies the maximum amount of characters in a request. All characters exceeding this limit are silently ignored.

`/ncs-config/webui/cgi/php`

PHP support

`/ncs-config/webui/cgi/php/enabled (boolean) [false]`

`enabled` is either 'true' or 'false'. If 'true', PHP support is enabled.

`/ncs-config/webui/idle-timeout (xs:duration) [PT30M]`

Maximum idle time before terminating a Web UI session. PT0M means no timeout. Default is PT30M, ie 30 minutes.

`/ncs-config/webui/absolute-timeout (xs:duration) [PT60M]`

Maximum absolute time before terminating a Web UI session. PT0M means no timeout. Default is PT60M, ie 60 minutes.

`/ncs-config/webui/rate-limiting (uint64) [1000000]`

Maximum number of allowed JSON-RPC requests every hour. 0 means infinity. Default is 1 million.

`/ncs-config/webui/audit (boolean) [false]`

`audit` is either 'true' or 'false'. If 'true', then JSON-RPC/CGI requests are logged to the audit log.

`/ncs-config/japi`

Java-API parameters.

`/ncs-config/japi/new-session-timeout (xs:duration) [PT30S]`

Timeout for a data provider to respond to a control socket request, see `DpTrans`. If the Dp fails to respond within the given time, it will be disconnected.

`/ncs-config/japi/query-timeout (xs:duration) [PT120S]`

Timeout for a data provider to respond to a worker socket query, see `DpTrans`. If the dp fails to respond within the given time, it will be disconnected.

`/ncs-config/japi/connect-timeout (xs:duration) [PT60S]`

Timeout for data provider to send initial message after connecting the socket to the NCS server. If the dp fails to initiate the connection within the given time, it will be disconnected.

```

/ncs-config/japi/object-cache-timeout (xs:duration) [PT2S]
    Timeout for the cache used by the getObject() and iterator(),nextObject() callback requests. NCS
    caches the result of these calls and serves getElem() requests from northbound agents from the cache.
    NOTE: Setting this timeout too low will effectively cause the callbacks to be non-functional - e.g.
    getObject() may be invoked for each getElem() request from a northbound agent.

/ncs-config/japi/event-reply-timeout (xs:duration) [PT120S]
    Timeout for the reply from an event notification subscriber for a notification that requires a reply, see
    the Notif class. If the subscriber fails to reply within the given time, the event notification socket will
    be closed.

/ncs-config/netconf-north-bound
    This section defines settings which decide how the NETCONF agent should behave, with respect to
    NETCONF and SSH.

/ncs-config/netconf-north-bound/enabled (boolean) [true]
    enabled is either 'true' or 'false'. If 'true', the NETCONF agent is started.

/ncs-config/netconf-north-bound/transport
    Settings deciding which transport services the NETCONF agent should listen on, e.g. TCP and SSH.

/ncs-config/netconf-north-bound/transport/ssh
    Settings deciding how the NETCONF SSH transport service should behave.

/ncs-config/netconf-north-bound/transport/ssh/enabled (boolean) [true]
    enabled is either 'true' or 'false'. If 'true', the NETCONF agent uses SSH as a transport service.

/ncs-config/netconf-north-bound/transport/ssh/ip (ipv4-address | ipv6-
address) [0.0.0.0]
    ip is an IP address which the NCS NETCONF agent should listen on. 0.0.0.0 means that it listens on
    the port (/ncs-config/netconf-north-bound/transport/ssh/port) for all IPv4 addresses on the machine.

/ncs-config/netconf-north-bound/transport/ssh/port (port-number) [2022]
    port is a valid port number to be used in combination with /ncs-config/netconf-north-bound/transport/
    ssh/ip. Note that the standard port for NETCONF over SSH is 830.

/ncs-config/netconf-north-bound/transport/ssh/extra-listen
    A list of additional IP address and port pairs which the NCS NETCONF agent should also listen on.

/ncs-config/netconf-north-bound/transport/ssh/extra-listen/ip (ipv4-
address | ipv6-address)

/ncs-config/netconf-north-bound/transport/ssh/extra-listen/port (port-
number)

/ncs-config/netconf-north-bound/transport/tcp
    NETCONF over TCP is not standardized, but it can be useful during development in order to use e.g.
    netcat for scripting. It is also useful if we want to use our own proprietary transport. In that case we
    setup the NETCONF agent to listen on localhost and then proxy it from our transport service module.

/ncs-config/netconf-north-bound/transport/tcp/enabled (boolean) [false]
    enabled is either 'true' or 'false'. If 'true', the NETCONF agent uses clear text TCP as a transport
    service.

/ncs-config/netconf-north-bound/transport/tcp/ip (ipv4-address | ipv6-
address) [0.0.0.0]
    ip is an IP address which the NCS NETCONF agent should listen on. 0.0.0.0 means that it listens on
    the port (/ncs-config/netconf-north-bound/transport/tcp/port) for all IPv4 addresses on the machine.

```

`/ncs-config/netconf-north-bound/transport/tcp/port (port-number) [2023]`
port is a valid port number to be used in combination with `/ncs-config/netconf-north-bound/transport/tcp/ip`.

`/ncs-config/netconf-north-bound/transport/tcp/extra-listen`

A list of additional IP address and port pairs which the NCS NETCONF agent should also listen on.

`/ncs-config/netconf-north-bound/transport/tcp/extra-listen/ip (ipv4-address | ipv6-address)`

`/ncs-config/netconf-north-bound/transport/tcp/extra-listen/port (port-number)`

`/ncs-config/netconf-north-bound/extended-sessions (boolean) [false]`

If extended-sessions are enabled, all NCS sessions can be terminated using `<kill-session>`, i.e. not only can other NETCONF session be terminated, but also CLI sessions, Webui sessions etc. If such a session holds a lock, it's session id will be returned in the `<lock-denied>`, instead of '0'.

Strictly speaking, this extension is not covered by the NETCONF specification; therefore it's false by default.

`/ncs-config/netconf-north-bound/idle-timeout (xs:duration) [PT0S]`

Maximum idle time before terminating a NETCONF session. If the session is waiting for notifications, or has a pending confirmed commit, the idle timeout is not used. The default value is 0, which means no timeout.

Modification of this value will only affect connections that are established after the modification has been done.

`/ncs-config/netconf-north-bound/write-timeout (xs:duration) [PT0S]`

Maximum time for a write operation towards a client to complete. If the time is exceeded, the NETCONF session is terminated. The default value is 0, which means no timeout.

Modification of this value will only affect connections that are established after the modification has been done.

`/ncs-config/netconf-north-bound/rpc-errors (close | inline) [close]`

If `rpc-errors` is 'inline', and an error occurs during the processing of a `<get>` or `<get-config>` request when NCS tries to fetch some data from a data provider, NCS will generate an `rpc-error` element in the faulty element, and continue to process the next element.

If an error occurs and `rpc-errors` is 'close', the NETCONF transport is closed by NCS.

`/ncs-config/netconf-north-bound/max-batch-processes (uint32 | unbounded) [unbounded]`

Controls how many concurrent NETCONF batch processes there can be at any time. A batch process can be started by the agent if a new NETCONF operation is implemented as a batch operation. See the NETCONF chapter in the NCS User's Guide for details.

`/ncs-config/netconf-north-bound/capabilities`

Decide which NETCONF capabilities to enable here.

`/ncs-config/netconf-north-bound/capabilities/url`

Turn on the URL capability options we want to support.

`/ncs-config/netconf-north-bound/capabilities/url/enabled (boolean) [false]`

enabled is either 'true' or 'false'. If 'true', the url NETCONF capability is enabled.

/ncs-config/netconf-north-bound/capabilities/url/file

Decide how the url file support should behave.

/ncs-config/netconf-north-bound/capabilities/url/file/enabled (boolean)
[true]

enabled is either 'true' or 'false'. If 'true', the url file scheme is enabled.

/ncs-config/netconf-north-bound/capabilities/url/file/root-dir (string)

rootDir is a directory path on disk where ConfD will store the result from an NETCONF operation using the url capability. This parameter must be set if the file url scheme is enabled.

/ncs-config/netconf-north-bound/capabilities/url/ftp

Decide how the url ftp scheme should behave.

/ncs-config/netconf-north-bound/capabilities/url/ftp/enabled (boolean)
[true]

enabled is either 'true' or 'false'. If 'true', the url ftp scheme is enabled.

/ncs-config/netconf-north-bound/capabilities/url/sftp

Decide how the url sftp scheme should behave.

/ncs-config/netconf-north-bound/capabilities/url/sftp/enabled (boolean)
[true]

enabled is either 'true' or 'false'. If 'true', the url sftp scheme is enabled.

/ncs-config/netconf-north-bound/capabilities/inactive

Control of the inactive capability option.

/ncs-config/netconf-north-bound/capabilities/inactive/enabled (boolean)
[true]

enabled is either 'true' or 'false'. If 'true', the 'http://tail-f.com/ns/netconf/inactive/1.0' capability is enabled.

/ncs-config/southbound-source-address

This section provides the possibility to specify the source address to use for southbound connections from NCS to the devices. In most cases the source address assignment is best left to the TCP/IP stack in the OS, since an incorrectly chosen address may result in connection failures. However in case there is more than one address that could be chosen by the stack, and we need to restrict the choice to one of them, these settings can be used.

/ncs-config/southbound-source-address/ipv4 (ipv4-address)

The source address to use for southbound IPv4 connections. If not set, the source address will be assigned by the OS.

/ncs-config/southbound-source-address/ipv6 (ipv6-address)

The source address to use for southbound IPv6 connections. If not set, the source address will be assigned by the OS.

/ncs-config/ha

/ncs-config/ha/enabled (boolean) [false]

If set to true, the HA mode is enabled.

/ncs-config/ha/ip (ipv4-address | ipv6-address) [0.0.0.0]

The IP address which NCS listens to for incoming connections from other HA nodes

/ncs-config/ha/port (port-number) [4570]

The port number which NCS listens to for incoming connections from other HA nodes

`/ncs-config/ha/tick-timeout (xs:duration) [PT20S]`

Defines the timeout between keepalive ticks sent between HA nodes. The special value 'PT0' means that no keepalive ticks will ever be sent.

`/ncs-config/scripts`

It is possible to add scripts to control various things in NCS, such as post-commit callbacks. New CLI commands can also be added. The scripts must be stored under `/ncs-config/scripts/dir` where there is a sub-directory for each script category. For some script categories it suffices to just add a script in the correct the sub-directory in order to enable the script. For others some configuration needs to be done.

`/ncs-config/scripts/dir (string)`

This parameter may be given multiple times.

Directory path to the location of plug-and-play scripts. The scripts directory must have the following sub-directories:

`scripts/command/ post-commit/`

`/ncs-config/large-scale`

`/ncs-config/large-scale/lsa`

`/ncs-config/large-scale/lsa/enabled (boolean) [false]`

Enable Layered Service Architecture, LSA. This requires a separate Cisco Smart License.

`/ncs-config/java-vm`

This section provides the possibility to override parameters in the `tailf-ncs-java-vm.yang` submodule, thus preventing setting of those parameters via northbound interfaces from having any effect, even if the NACM access rules allow it.

Refer to `tailf-ncs-java-vm.yang` for a detailed description of the parameters.

`/ncs-config/java-vm/start-command (string)`

The command which NCS will run to start the Java VM, or the string `DEFAULT`. Setting a value here overrides any setting for `/java-vm/start-command` in the `tailf-ncs-java-vm.yang` submodule. The string `DEFAULT` means that the default as specified in `tailf-ncs-java-vm.yang` should be used.

`/ncs-config/java-vm/run-in-terminal`

`/ncs-config/java-vm/run-in-terminal/terminal-command (string)`

The command which NCS will run to start the terminal, or the string `DEFAULT`. Setting a value here overrides any setting for `/java-vm/run-in-terminal/terminal-command` in the `tailf-ncs-java-vm.yang` submodule. The string `DEFAULT` means that the default as specified in `tailf-ncs-java-vm.yang` should be used.

`/ncs-config/python-vm`

This section provides the possibility to override parameters in the `tailf-ncs-python-vm.yang` submodule, thus preventing setting of those parameters via northbound interfaces from having any effect, even if the NACM access rules allow it.

Refer to `tailf-ncs-python-vm.yang` for a detailed description of the parameters.

`/ncs-config/python-vm/start-command (string)`

The command which NCS will run to start the Python VM, or the string `DEFAULT`. Setting a value here overrides any setting for `/python-vm/start-command` in the `tailf-ncs-python-vm.yang` submodule.

The string `DEFAULT` means that the default as specified in `tailf-ncs-python-vm.yang` should be used.

`/ncs-config/python-vm/run-in-terminal`

`/ncs-config/python-vm/run-in-terminal/terminal-command (string)`

The command which NCS will run to start the terminal, or the string DEFAULT. Setting a value here overrides any setting for `/python-vm/run-in-terminal/terminal-command` in the `tailf-ncs-python-vm.yang` submodule. The string DEFAULT means that the default as specified in `tailf-ncs-python-vm.yang` should be used.

`/ncs-config/python-vm/logging`

`/ncs-config/python-vm/logging/log-file-prefix (string)`

The prefix used for the Python VM log file, or the string DEFAULT. Setting a value here overrides any setting for `/python-vm/logging/log-file-prefix` in the `tailf-ncs-python-vm.yang` submodule. The string DEFAULT means that the default as specified in `tailf-ncs-python-vm.yang` should be used.

`/ncs-config/smart-license`

This section provides the possibility to override parameters in the `tailf-ncs-smart-license.yang` submodule, thus preventing setting of those parameters via northbound interfaces from having any effect, even if the NACM access rules allow it.

Refer to `tailf-ncs-smart-license.yang` for a detailed description of the parameters.

`/ncs-config/smart-license/smart-agent`

`/ncs-config/smart-license/smart-agent/java-executable (string)`

The Java VM executable that NCS will use for smart licensing, or the string DEFAULT. Setting a value here overrides any setting for `/smart-license/smart-agent/java-executable` in the `tailf-ncs-smart-license.yang` submodule. The string DEFAULT means that the default as specified in `tailf-ncs-smart-license.yang` should be used.

`/ncs-config/smart-license/smart-agent/java-options (string)`

Options which NCS will use when starting the Java VM, or the string DEFAULT. Setting a value here overrides any setting for `/smart-license/smart-agent/java-options` in the `tailf-ncs-smart-license.yang` submodule. The string DEFAULT means that the default as specified in `tailf-ncs-smart-license.yang` should be used.

`/ncs-config/smart-license/smart-agent/proxy`

`/ncs-config/smart-license/smart-agent/proxy/url (uri | string)`

Proxy URL for the smart licensing agent, or the string DEFAULT. Setting a value here overrides any setting for `/smart-license/smart-agent/proxy/url` in the `tailf-ncs-smart-license.yang` submodule. The string DEFAULT effectively disables the proxy URL, since there is no default specified in `tailf-ncs-smart-license.yang`.

YANG TYPES

bsd-facility-type

The facility argument is used to specify what type of program is logging the message. This lets the syslog configuration file specify that messages from different facilities will be handled differently

cli-max

completion-meta-info-type

crypt-hash-algorithm-type

crypt-hash-rounds-type

dh-group-size-type

hex16-value-type

hex8-value-type

infinity-type

ip-address

ipv4-address

ipv6-address

limit-type

object-identifier

port-number

syslog-facility-type

syslog-version-type

timeout-type

unbounded-type

uri

SEE ALSO

ncs(1) - command to start and control the NCS daemon

Name

tailf_yang_cli extensions — Tail-f YANG CLI extensions

Synopsis

tailf:cli-add-mode
tailf:cli-allow-join-with-key
tailf:cli-allow-join-with-value
tailf:cli-allow-key-abbreviation
tailf:cli-allow-range
tailf:cli-allow-wildcard
tailf:cli-autowizard
tailf:cli-boolean-no
tailf:cli-break-sequence-commands
tailf:cli-case-insensitive
tailf:cli-case-sensitive
tailf:cli-column-align
tailf:cli-column-stats
tailf:cli-column-width
tailf:cli-compact-stats
tailf:cli-compact-syntax
tailf:cli-completion-actionpoint
tailf:cli-configure-mode
tailf:cli-custom-error
tailf:cli-custom-range
tailf:cli-custom-range-actionpoint
tailf:cli-custom-range-enumerator
tailf:cli-delayed-auto-commit
tailf:cli-delete-container-on-delete
tailf:cli-delete-when-empty
tailf:cli-diff-after
tailf:cli-diff-before
tailf:cli-diff-create-after

tailf:cli-diff-create-before
tailf:cli-diff-delete-after
tailf:cli-diff-delete-before
tailf:cli-diff-dependency
tailf:cli-diff-modify-after
tailf:cli-diff-modify-before
tailf:cli-diff-set-after
tailf:cli-diff-set-before
tailf:cli-disabled-info
tailf:cli-disallow-value
tailf:cli-display-empty-config
tailf:cli-display-separated
tailf:cli-drop-node-name
tailf:cli-embed-no-on-delete
tailf:cli-enforce-table
tailf:cli-exit-command
tailf:cli-explicit-exit
tailf:cli-expose-key-name
tailf:cli-expose-ns-prefix
tailf:cli-flat-list-syntax
tailf:cli-flatten-container
tailf:cli-full-command
tailf:cli-full-no
tailf:cli-full-show-path
tailf:cli-hide-in-submode
tailf:cli-ignore-modified
tailf:cli-incomplete-command
tailf:cli-incomplete-no
tailf:cli-incomplete-show-path
tailf:cli-instance-info-leafs
tailf:cli-key-format

tailf:cli-list-syntax
tailf:cli-min-column-width
tailf:cli-mode-name
tailf:cli-mode-name-actionpoint
tailf:cli-mount-point
tailf:cli-multi-line-prompt
tailf:cli-multi-value
tailf:cli-multi-word-key
tailf:cli-no-key-completion
tailf:cli-no-keyword
tailf:cli-no-match-completion
tailf:cli-no-name-on-delete
tailf:cli-no-value-on-delete
tailf:cli-oper-info
tailf:cli-operational-mode
tailf:cli-optional-in-sequence
tailf:cli-prefix-key
tailf:cli-preformatted
tailf:cli-range-delimiters
tailf:cli-range-list-syntax
tailf:cli-recursive-delete
tailf:cli-remove-before-change
tailf:cli-replace-all
tailf:cli-reset-container
tailf:cli-run-template
tailf:cli-run-template-enter
tailf:cli-run-template-footer
tailf:cli-run-template-legend
tailf:cli-sequence-commands
tailf:cli-show-config
tailf:cli-show-long-obu-diffs
tailf:cli-show-no

```
tailf:cli-show-obu-comments
tailf:cli-show-order-tag
tailf:cli-show-order-taglist
tailf:cli-show-template
tailf:cli-show-template-enter
tailf:cli-show-template-footer
tailf:cli-show-template-legend
tailf:cli-show-with-default
tailf:cli-strict-leafref
tailf:cli-suppress-key-abbreviation
tailf:cli-suppress-key-sort
tailf:cli-suppress-list-no
tailf:cli-suppress-mode
tailf:cli-suppress-no
tailf:cli-suppress-quotes
tailf:cli-suppress-range
tailf:cli-suppress-shortenabled
tailf:cli-suppress-show-conf-path
tailf:cli-suppress-show-match
tailf:cli-suppress-show-path
tailf:cli-suppress-silent-no
tailf:cli-suppress-table
tailf:cli-suppress-validation-warning-prompt
tailf:cli-suppress-warning
tailf:cli-suppress-wildcard
tailf:cli-table-footer
tailf:cli-table-legend
tailf:cli-trim-default
tailf:cli-value-display-template
```

DESCRIPTION

This manpage describes all the Tail-f CLI extension statements.

The YANG source file `$NCS_DIR/src/ncs/yang/tailf-cli-extensions.yang` gives the exact YANG syntax for all Tail-f YANG CLI extension statements - using the YANG language itself.

Most of the concepts implemented by the extensions listed below are described in the User Guide.

YANG STATEMENTS

tailf:cli-add-mode

Creates a mode of the container.

Can be used in config nodes only.

Used in I- and C-style CLIs.

The *cli-add-mode* statement can be used in: *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-allow-join-with-key

Indicates that the list name may be written together with the first key, without requiring a whitespace in between, ie allowing both interface ethernet1/1 and interface ethernet 1/1

Used in I- and C-style CLIs.

The *cli-allow-join-with-key* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-display-joined Specifies that the joined version should be used when displaying the configuration in C- and I- mode.

tailf:cli-allow-join-with-value

Indicates that the leaf name may be written together with the value, without requiring a whitespace in between, ie allowing both interface ethernet1/1 and interface ethernet 1/1

Used in I- and C-style CLIs.

The *cli-allow-join-with-value* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-display-joined Specifies that the joined version should be used when displaying the configuration in C- and I- mode.

tailf:cli-allow-key-abbreviation

Key values can be abbreviated.

In the J-style CLI this is relevant when using the commands 'delete' and 'edit'.

In the I- and C-style CLIs this is relevant when using the commands 'no', 'show configuration' and for commands to enter submodes.

See also `/confdConfig/cli/allowAbbrevKeys` in `confd.conf(5)`.

The *cli-allow-key-abbreviation* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-allow-range

Means that the non-integer key should allow range expressions.

Can be used in key leafs only.

Used in J-, I- and C-style CLIs.

The *cli-allow-range* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-allow-wildcard

Means that the list allows wildcard expressions in the 'show' pattern.

See also /confdConfig/cli/allowWildcard in confd.conf(5).

Used in J-, I- and C-style CLIs.

The *cli-allow-wildcard* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-autowizard

Specifies that the autowizard should include this leaf even if the leaf is optional.

One use case is when implementing pre-configuration of devices. A config false node can be defined for showing if the configuration is active or not (preconfigured).

Used in J-, I- and C-style CLIs.

The *cli-autowizard* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-boolean-no

Specifies that a leaf of type boolean should be displayed as '<leafname>' if set to true, and 'no <leafname>' if set to false.

Cannot be used in conjunction with tailf:cli-hide-in-submode or tailf:cli-compact-syntax.

Used in I- and C-style CLIs.

The *cli-boolean-no* statement can be used in: *typedef*, *leaf*, *refine*, and *tailf:symlink*.

The following substatements can be used:

tailf:cli-reversed Specified that true should be displayed as 'no <name>' and false as 'name'.

Used in I- and C-style CLIs.

tailf:cli-suppress-warning

tailf:cli-break-sequence-commands

Specifies that previous cli-sequence-command declaration should stop at this point. Only applicable when a cli-sequence-command declaration has been used in the parent container.

Used in I- and C-style CLIs.

The *cli-break-sequence-commands* statement can be used in: *leaf*, *tailf:symlink*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-case-insensitive

Specifies that node is case-insensitive. If applied to a container or a list, any nodes below will also be case-insensitive.

Node names are discovered without care of the case. Also affect matching of key values in lists. However it doesn't affect the storing of a leaf value. E.g. a modification of a leaf value from upper case to lower case is still considered a modification of data.

Note that this will override any case-insensitivity settings configured in *confd.conf*

The *cli-case-insensitive* statement can be used in: *container*, *list*, and *leaf*.

tailf:cli-case-sensitive

Specifies that this node is case-sensitive. If applied to a container or a list, any nodes below will also be case-sensitive.

This negates the *cli-case-insensitive* extension (see below).

Note that this will override any case-sensitivity settings configured in *confd.conf*

The *cli-case-sensitive* statement can be used in: *container*, *list*, and *leaf*.

tailf:cli-column-align *value*

Specifies the alignment of the data in the column in the auto-rendered tables.

Used in J-, I- and C-style CLIs.

The *cli-column-align* statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-column-stats

Display leafs in the container as columns, i.e., do not repeat the name of the container on each line, but instead indent each leaf under the container.

Used in I- and C-style CLIs.

The *cli-column-stats* statement can be used in: *container*, *refine*, and *tailf:symlink*.

tailf:cli-column-width *value*

Set a fixed width for the column in the auto-rendered tables.

Used in J-, I- and C-style CLIs.

The *cli-column-width* statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-compact-stats

Instructs the CLI engine to use the compact representation for this node. The compact representation means that all leaf elements are shown on a single line.

Used in J-, I- and C-style CLIs.

The *cli-compact-stats* statement can be used in: *list*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-wrap If present, the line will be wrapped at screen width.

tailf:cli-width Specifies a fixed terminal width to use before wrapping line. It is only used when *tailf:cli-wrap* is present. If a width is not specified the line is wrapped when the terminal width is reached.

tailf:cli-delimiter Specifies a string to print between the leaf name and its value when displaying leaf values.

tailf:cli-prettyfy If present, dashes (-) and underscores (_) in leaf names are replaced with spaces.

tailf:cli-spacer Specifies a string to print between the nodes.

tailf:cli-compact-syntax

Instructs the CLI engine to use the compact representation for this node in the 'show running-configuration' command. The compact representation means that all leaf elements are shown on a single line.

Cannot be used in conjunction with *tailf:cli-boolean-no*.

Used in I- and C-style CLIs.

The *cli-compact-syntax* statement can be used in: *list*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-completion-actionpoint value

Specifies that completion for the leaf values is done through a callback function.

The argument is the name of an actionpoint, which must be implemented by custom code. In the actionpoint, the `completion()` callback function will be invoked. See `confd_lib_dp(3)` for details.

Used in J-, I- and C-style CLIs.

The *cli-completion-actionpoint* statement can be used in: *leaf-list*, *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-completion-id Specifies a string which is passed to the callback when invoked. This makes it possible to use the same callback at several locations and still keep track of which point it is invoked from.

tailf:cli-configure-mode

An action or rpc with this attribute will be available in configure mode, but not in operational mode.

The default is that the action or rpc is available in both configure and operational mode.

Used in J-, I- and C-style CLIs.

The *cli-configure-mode* statement can be used in: *tailf:action* and *rpc*.

tailf:cli-custom-error text

This statement specifies a custom error message to be displayed when the user enters an invalid value.

The *cli-custom-error* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-custom-range

Specifies that the key should support ranges. A type matching the range expression must be supplied.

Can be used in key leafs only.

Used in J-, I- and C-style CLIs.

The *cli-custom-range* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-range-type This statement contains the name of a derived type, possibly with a prefix. If no prefix is given, the type must be defined in the local module. For example:

```
cli-range-type p:my-range-type;
```

All range expressions must match this type, and a valid key value must not match this type.

tailf:cli-suppress-warning

tailf:cli-custom-range-actionpoint value

Specifies that the list supports range expressions and that a custom function will be invoked to determine if an instance belong in the range or not. At least one key element needs a cli-custom-range statement.

The argument is the name of an actionpoint, which must be implemented by custom code. In the actionpoint, the completion() callback function will be invoked. See confd_lib_dp(3) for details.

When a range expression value which matches the type is given in the CLI, the CLI engine will invoke the callback with each existing list entry instance. If the callback returns CONFD_OK, it matches the range expression, and if it returns CONFD_ERR, it doesn't match.

Used in J-, I- and C-style CLIs.

The *cli-custom-range-actionpoint* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-completion-id Specifies a string which is passed to the callback when invoked. This makes it possible to use the same callback at several locations and still keep track of which point it is invoked from.

tailf:cli-allow-caching Allow caching of the evaluation results between different parent paths.

tailf:cli-suppress-warning

tailf:cli-custom-range-enumerator value

Specifies a callback to invoke to get an array of instances matching a regular expression. This is used when instances should be allowed to be created using a range expression in set.

The callback is not used for delete or show operations.

The callback is allowed to return a superset of all matching instances since the instances will be filtered using the range expression afterwards.

Used in J-, I- and C-style CLIs.

The *cli-custom-range-enumerator* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-completion-id Specifies a string which is passed to the callback when invoked. This makes it possible to use the same callback at several locations and still keep track of which point it is invoked from.

tailf:cli-allow-caching Allow caching of the evaluation results between different parent paths.

tailf:cli-suppress-warning

tailf:cli-delayed-auto-commit

Enables transactions while in a specific submode (or submode of that mode). The modifications performed in that mode will not take effect until the user exits that submode.

Can be used in config nodes only. If used in a container, the container must also have a *tailf:cli-add-mode* statement, and if used in a list, the list must not also have a *tailf:cli-suppress-mode* statement.

Used in I- and C-style CLIs.

The *cli-delayed-auto-commit* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-delete-container-on-delete

Specifies that the parent container should be deleted when . this leaf is deleted.

The *cli-delete-container-on-delete* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-delete-when-empty

Instructs the CLI engine to delete the list when the last list instance is deleted'. Requires that *cli-suppress-mode* is set.

The behavior is recursive. If all optional leafs in a list instance are deleted the list instance itself is deleted. If that list instance happens to be the last list instance in a list it is also deleted. And so on. Used in I- and C-style CLIs.

The *cli-delete-when-empty* statement can be used in: *list* and *container*.

tailf:cli-diff-after *path*

When displaying C-style configuration diffs, display any changes made to this node after any changes made to the target node(s).

Thus, the dependency will trigger when any changes (created, modified or deleted) has been made to this node while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-after* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-before *path*

When displaying C-style configuration diffs, display any changes made to this node before any changes made to the target node(s).

Thus, the dependency will trigger when any changes (created, modified or deleted) has been made to this node while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-before* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-create-after *path*

When displaying C-style configuration diffs, display any create operations made on this node after any changes made to the target node(s).

Thus, the dependency will trigger when this node has been created while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-create-after* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-create-before *path*

When displaying C-style configuration diffs, display any create operations made on this node before any changes made to the target node(s).

Thus, the dependency will trigger when this node has been created while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-create-before* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-delete-after *path*

When displaying C-style configuration diffs, display any delete operations made on this node after any changes made to the target node(s).

Thus, the dependency will trigger when this node has been deleted while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-delete-after* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-delete-before *path*

When displaying C-style configuration diffs, display any delete operations made on this node before any changes made to the target node(s).

Thus, the dependency will trigger when this node has been deleted while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-delete-before* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-dependency *path*

Tells the 'show configuration' command, and the diff generator that this node depends on another node. When removing the node with this declaration, it should be removed before the node it depends on is removed, ie the declaration controls the ordering of the commands in the 'show configuration' output.

Applies to C-style

The *cli-diff-dependency* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-trigger-on-set Specify that the dependency should trigger on set/modify of the target path, but deletion of the target will trigger the current node to be placed in front of the target.

The annotation can be used to get the diff behavior where one leaf is first deleted before the other leaf is set. For example, having the data model below:

```
container X { leaf A { tailf:cli-diff-dependency "../B" { tailf:cli-trigger-on-set; } type empty; } leaf B { tailf:cli-diff-dependency "../A" { tailf:cli-trigger-on-set; } type empty; } }
```

produces the following diffs when setting one leaf and deleting the other

no X A X B

and

no X B X A

this can also be done with list instances, for example

```
list a { key id;
leaf id { tailf:cli-diff-dependency "/c[id=current()../id]" { tailf:cli-trigger-on-set; } type string; } }
list c { key id; leaf id { tailf:cli-diff-dependency "/a[id=current()../id]" { tailf:cli-trigger-on-set; } type
string; } }
```

we get

```
no a foo c foo !
```

and

```
no c foo a foo !
```

In the above case if we have the same id in list "a" and "c" and we delete the instance in one list, and add it in the other, then the deletion will always precede the create.

tailf:cli-trigger-on-delete This annotation can be used together with *tailf:cli-trigger-on-set* to also get the behavior that when deleting the target display changes to this node first. For example:

```
container settings { tailf:cli-add-mode;
leaf opmode { tailf:cli-no-value-on-delete;
type enumeration { enum nat; enum transparent; } }
leaf manageip { when "../opmode = 'transparent'"; mandatory true; tailf:cli-no-value-on-delete; tailf:cli-
diff-dependency '../opmode' { tailf:cli-trigger-on-set; tailf:cli-trigger-on-delete; }
type string; } }
```

What we are trying to achieve here is that if *manageip* is deleted, it should be displayed before *opmode*, but if we configure both *opmode* and *manageip*, we should display *opmode* first, ie get the diffs:

```
settings opmode transparent manageip 1.1.1.1 !
```

and

```
settings no manageip opmode nat !
```

and

```
settings no manageip no opmode !
```

The *cli-trigger-on-set* annotation will cause the 'no *manageip*' command to be displayed before setting *opmode*. The *tailf:cli-trigger-on-delete* will cause 'no *manageip*' to be placed before 'no *opmode*' when both are deleted.

In the first diff where both are created, *opmode* will come first due to the *diff-dependency* setting, regardless of the *cli-trigger-on-delete* and *cli-trigger-on-set*.

tailf:cli-trigger-on-all Specify that the dependency should always trigger. It is the same as placing one element before another in the data model. For example, given the data model:

```
container X { leaf A { tailf:cli-diff-dependency '../B' { tailf:cli-trigger-on-all; } type empty; } leaf B { type
empty; } }
```

We get the diffs

X B X A

and

no X B no X A

tailf:cli-suppress-warning

tailf:cli-diff-modify-after *path*

When displaying C-style configuration diffs, display any modify operations made on this node after any changes made to the target node(s).

Thus, the dependency will trigger when this node has been modified (not created or deleted) while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-modify-after* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-modify-before *path*

When displaying C-style configuration diffs, display any modify operations made on this node before any changes made to the target node(s).

Thus, the dependency will trigger when this node has been modified (not created or deleted) while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-modify-before* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-set-after *path*

When displaying C-style configuration diffs, display any set operations (created or modified) made on this node after any changes made to the target node(s).

Thus, the dependency will trigger when this node has been set (created or modified) while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-set-after* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-diff-set-before *path*

When displaying C-style configuration diffs, display any set operations (created or modified) made on this node before any changes made to the target node(s).

Thus, the dependency will trigger when this node has been set (created or modified) while any changes (created, modified or deleted) has been made to the target node(s).

Applies to C-style

The *cli-diff-set-before* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:cli-when-target-set Specify that the dependency should trigger when the target node(s) has been set (created or modified). Note; using this sub-statement is equivalent with using both *tailf:cli-when-target-create* and *tailf:cli-when-target-modify*

tailf:cli-when-target-create Specify that the dependency should trigger when the target node(s) has been created

tailf:cli-when-target-modify Specify that the dependency should trigger when the target node(s) has been modified (not created or deleted)

tailf:cli-when-target-delete Specify that the dependency should trigger when the target node(s) has been deleted

tailf:cli-suppress-warning

tailf:cli-disabled-info value

Specifies an info string that will be used as a descriptive text for the value 'disable' (false) of boolean-typed leafs when the `confd.conf(5)` setting `/confdConfig/cli/useShortEnabled` is set to 'true'.

Used in J-, I- and C-style CLIs.

The *cli-disabled-info* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-disallow-value value

Specifies that a pattern for invalid values.

Used in I- and C-style CLIs.

The *cli-disallow-value* statement can be used in: *leaf*, *leaf-list*, *refine*, and *tailf:symlink*.

tailf:cli-display-empty-config

Specifies that the node will be included when doing a 'show stats', even if it is a non-config node, provided that the list contains at least one non-config node.

Used in J-style CLI.

The *cli-display-empty-config* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-display-separated

Tells CLI engine to display this container as a separate line item even when it has children. Only applies to presence containers.

Applicable for optional containers in the C- and I- style CLIs.

The *cli-display-separated* statement can be used in: *container*, *tailf:symlink*, and *refine*.

tailf:cli-drop-node-name

Specifies that the name of a node is not present in the CLI.

If *tailf:cli-drop-node-name* is given on a child to a list node, we recommend that you also use *tailf:cli-suppress-mode* on that list node, otherwise the CLI will be very confusing.

For example, consider this data model, from the *tailf-aaa* module:

```
list alias {
  key name;
  leaf name {
    type string;
  }
  leaf expansion {
    type string;
    mandatory true;
    tailf:cli-drop-node-name;
  }
}
```

```
}
```

If you type 'alias foo' in the CLI, you would end up in the 'alias' submenu. But since the expansion is dropped, you would end up specifying the expansion value without typing any command.

If, on the other hand, the 'alias' list had a `tailf:cli-suppress-mode` statement, you would set an expansion 'bar' by typing 'alias foo bar'.

`tailf:cli-drop-node-name` cannot be used inside `tailf:action`.

Used in I- and C-style CLIs.

The `cli-drop-node-name` statement can be used in: *leaf*, *container*, *list*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-embed-no-on-delete

Embed no in front of the element name instead of at the beginning of the line.

Applies to C-style

The `cli-embed-no-on-delete` statement can be used in: *leaf*, *container*, *list*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-enforce-table

Forces the generation of a table for a list element node regardless of whether the table will be too wide or not. This applies to the tables generated by the auto-rendered show commands for non-config data.

Used in I- and C-style CLIs.

The `cli-enforce-table` statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-exit-command *value*

Tells the CLI to add an explicit exit-from-submode command. The `tailf:info` substatement can be used for adding a custom info text for the command.

Used in I- and C-style CLIs.

The `cli-exit-command` statement can be used in: *list*, *container*, *refine*, and *tailf:symlink*.

The following substatements can be used:

tailf:info

tailf:cli-explicit-exit

Tells the CLI to add an explicit exit command when displaying the configuration. It will not be added if `cli-exit-command` is defined as well. The annotation is inherited by all sub-modes.

Used in I- and C-style CLIs.

The `cli-explicit-exit` statement can be used in: *list*, *container*, *refine*, and *tailf:symlink*.

tailf:cli-expose-key-name

Force the user to enter the name of the key and display the key name when displaying the running-configuration.

Used in J-, I- and C-style CLIs.

The *cli-expose-key-name* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-expose-ns-prefix

When used force the CLI to display namespace prefix of all children.

The *cli-expose-ns-prefix* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-flat-list-syntax

Specifies that elements in a leaf-list should be entered without surrounding brackets. Also, multiple elements can be added to a list or deleted from a list.

Used in J-, I- and C-style CLIs.

The *cli-flat-list-syntax* statement can be used in: *leaf-list* and *refine*.

The following substatements can be used:

tailf:cli-replace-all

tailf:cli-flatten-container

Allows the CLI to exit the container and continue to input from the parent container when all leaves in the current container has been set.

Can be used in config nodes only.

Used in I- and C-style CLIs.

The *cli-flatten-container* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-full-command

Specifies that an auto-rendered command should be considered complete, ie, no additional leaves or containers can be entered on the same command line.

Used in I- and C-style CLIs.

The *cli-full-command* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-full-no

Specifies that an auto-rendered 'no'-command should be considered complete, ie, no additional leaves or containers can be entered on the same command line.

Used in I- and C-style CLIs.

The *cli-full-no* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

tailf:cli-full-show-path

Specifies that a path to the show command is considered complete, i.e., no more elements can be added to the path. It can also be used to specify a maximum number of keys to be given for lists.

Used in J-, I- and C-style CLIs.

The *cli-full-show-path* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-max-keys Specifies the maximum number of allowed keys for the show command.

tailf:cli-hide-in-submode

Hide leaf when submode has been entered. Mostly useful when leaf has to be entered in order to enter a submode. Also works for flattened containers.

Cannot be used in conjunction with *tailf:cli-boolean-no*.

Used in I- and C-style CLIs.

The *cli-hide-in-submode* statement can be used in: *leaf*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-ignore-modified

Tells the *cdb_cli_diff_iterate* system call to not generate a CLI string when this container is modified. The string will instead be generated for the modified sub-element.

Applies to C-style and I-style

The *cli-ignore-modified* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-incomplete-command

Specifies that an auto-rendered command should be considered incomplete. Can be used to prevent <cr> from appearing in the completion list for optional internal nodes, for example, or to ensure that the user enters all leaf values in a container (if used in combination with *cli-sequence-commands*).

Used in I- and C-style CLIs.

The *cli-incomplete-command* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-incomplete-no

Specifies that an auto-rendered 'no'-command should not be considered complete, ie, additional leaves or containers must be entered on the same command line.

Used in I- and C-style CLIs.

The *cli-incomplete-no* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, and *refine*.

tailf:cli-incomplete-show-path

Specifies that a path to the show command is considered incomplete, i.e., it needs more elements added to the path. It can also be used to specify a minimum number of keys to be given for lists.

Used in J-, I- and C-style CLIs.

The *cli-incomplete-show-path* statement can be used in: *leaf*, *tailf:symlink*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-min-keys Specifies the minimum number of required keys for the show command.

tailf:cli-instance-info-leafs *value*

This statement is used to specify how list entries are displayed when doing completion in the CLI. By default, a list entry is displayed by listing its key values, and the value of a leaf called 'description', if such a leaf exists in the list entry.

The 'cli-instance-info-leafs' statement takes as its argument a space separated string of leaf names. When a list entry is displayed, the values of these leafs are concatenated with a space character as separator and shown to the user.

For example, when asked to specify an interface the CLI will display a list of possible interface instances, say 1 2 3 4. If the cli-instance-info-leafs property is set to 'description' then the CLI might show:

Possible completions: 1 - internet 2 - lab 3 - dmz 4 - wlan

Used in J-, I- and C-style CLIs.

The *cli-instance-info-leafs* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-key-format *value*

The format string is used when parsing a key value and when generating a key value for an existing configuration. The key items are numbered from 1-N and the format string should indicate how they are related by using \$(X) (where X is the key number). For example:

tailf:cli-key-format '\$(1)-\$(2)' means that the first key item is concatenated with the second key item by a '-'.

Used in J-, I- and C-style CLIs.

The *cli-key-format* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-list-syntax

Specifies that each entry in a leaf-list should be displayed as a separate element.

Used in J-, I- and C-style CLIs.

The *cli-list-syntax* statement can be used in: *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-multi-word Specifies that a multi-word value may be entered without quotes.

tailf:cli-min-column-width *value*

Set a minimum width for the column in the auto-rendered tables.

Used in J-, I- and C-style CLIs.

The *cli-min-column-width* statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-mode-name *value*

Specifies a custom mode name, instead of the default which is the name of the list or container node.

Can be used in config nodes only. If used in a container, the container must also have a tailf:cli-add-mode statement, and if used in a list, the list must not also have a tailf:cli-suppress-mode statement.

Variables for the list keys in the current mode are available. For examples, 'config-foo-xx \$(name)' (provided the key leaf is called 'name').

Used in I- and C-style CLIs.

The *cli-mode-name* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-mode-name-actionpoint *value*

Specifies that a custom function will be invoked to find out the mode name, instead of using the default with is the name of the list or container node.

The argument is the name of an actionpoint, which must be implemented by custom code. In the actionpoint, the command() callback function will be invoked, and it must return a string with the mode name. See confd_lib_dp(3) for details.

Can be used in config nodes only. If used in a container, the container must also have a tailf:cli-add-mode statement, and if used in a list, the list must not also have a tailf:cli-suppress-mode statement.

Used in I- and C-style CLIs.

The *cli-mode-name-actionpoint* statement can be used in: *container*, *list*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-mount-point *value*

By default actions are mounted under the 'request' command in the J-style CLI and at the top-level in the I- and C-style CLIs. This annotation allows the action to be mounted under other top level commands

The *cli-mount-point* statement can be used in: *tailf:action* and *rpc*.

tailf:cli-multi-line-prompt

Tells the CLI to automatically enter multi-line mode when prompting the user for a value to this leaf.

Used in J-, I- and C-style CLIs.

The *cli-multi-line-prompt* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-multi-value

Specifies that all remaining tokens on the command line should be considered a value for this leaf. This prevents the need for quoting values containing spaces, but also prevents multiple leaves from being set on the same command line once a multi-value leaf has been given on a line.

If the tailf:cli-max-words substatements is used then additional leaves may be entered.

Note: This extension isn't applicable in actions

Used in I- and C-style CLIs.

The *cli-multi-value* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-max-words Specifies the maximum number of allowed words for the key or value.

tailf:cli-multi-word-key

Specifies that the key should allow multiple tokens for the value. Proper type restrictions needs to be used to limit the range of the leaf value.

Can be used in key leafs only.

Note: This extension isn't applicable in actions

Used in J-, I- and C-style CLIs.

The *cli-multi-word-key* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-max-words Specifies the maximum number of allowed words for the key or value.

tailf:cli-no-key-completion

Specifies that the CLI engine should not perform completion for key leafs in the list. This is to avoid querying the data provider for all existing keys.

Used in J-, I- and C-style CLIs.

The *cli-no-key-completion* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-no-keyword

Specifies that the name of a node is not present in the CLI.

Note that is must be used with some care, just like *tailf:cli-drop-node-name*. The resulting data model must still be possible to parse deterministically. For example, consider the data model

```
container interfaces {
  list traffic {
    tailf:cli-no-keyword;
    key id;
    leaf id { type string; }
    leaf mtu { type uint16; }
  }
  list management {
    tailf:cli-no-keyword;
    key id;
    leaf id { type string; }
    leaf mtu { type uint16; }
  }
}
```

In this case it is impossible to determine if the config

```
interfaces {
  eth0 {
```

```

        mtu 1400;
    }
}

```

Means that there should be an traffic interface instance named 'eth0' or a management interface instance named 'eth0'. If, on the other hand, a restriction on the type was used, for example

```

container interfaces {
    list traffic {
        tailf:cli-no-keyword;
        key id;
        leaf id { type string; pattern 'eth.*'; }
        leaf mtu { type uint16; }
    }
    list management {
        tailf:cli-no-keyword;
        key id;
        leaf id { type string; pattern 'lo.*'; }
        leaf mtu { type uint16; }
    }
}

```

then the problem would disappear.

Used in the J-style CLIs.

The *cli-no-keyword* statement can be used in: *leaf*, *container*, *list*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-no-match-completion

Specifies that the CLI engine should not provide match completion for the key leafs in the list.

Used in J-, I- and C-style CLIs.

The *cli-no-match-completion* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-no-name-on-delete

When displaying the deleted version of this element do not include the name.

Applies to C-style

The *cli-no-name-on-delete* statement can be used in: *leaf*, *container*, *list*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-no-value-on-delete

When displaying the deleted version of this leaf do not include the old value.

Applies to C-style

The *cli-no-value-on-delete* statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-oper-info text

This statement works exactly as *tailf:info*, with the exception that it is used when displaying the element info in the context of stats.

Both *tailf:info* and *tailf:cli-oper-info* can be present at the same time.

The *cli-oper-info* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *rpc*, *identity*, *tailf:action*, *tailf:symlink*, and *refine*.

tailf:cli-operational-mode

An action or rpc with this attribute will be available in operational mode, but not in configure mode.

The default is that the action or rpc is available in both configure and operational mode.

Used in J-, I- and C-style CLIs.

The *cli-operational-mode* statement can be used in: *tailf:action* and *rpc*.

tailf:cli-optional-in-sequence

Specifies that this element is optional in the sequence. If it is set it must be set in the right sequence but may be skipped.

Used in I- and C-style CLIs.

The *cli-optional-in-sequence* statement can be used in: *leaf*, *tailf:symlink*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-prefix-key

This leaf has to be given as a prefix before entering the actual list keys. Very backwards but a construct that exists in some Cisco CLIs.

The construct can be used also for leaf-lists but only when then *tailf:cli-range-list-syntax* is also used.

Used in I- and C-style CLIs.

The *cli-prefix-key* statement can be used in: *leaf*, *tailf:symlink*, *refine*, and *leaf-list*.

The following substatements can be used:

tailf:cli-before-key Specifies before which key the prefix element should be inserted. The first key has number 1.

tailf:cli-suppress-warning

tailf:cli-preformatted

Suppresses quoting of non-config elements when displaying them. Newlines will be preserved in strings etc.

Used in J-, I- and C-style CLIs.

The *cli-preformatted* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-range-delimiters value

Allows for custom delimiters to be defined for range expressions. By default only / is considered a delimiter, ie when processing a key like 1/2/3 then each of 1, 2 and 3 will be matched separately against range expressions, ie given the expression 1-3/5-6/7,8 1 will be matched with 1-3, 2 with 5-6, and 3 with 7,8. If, for example, the delimiters value is set to './' then both '.' and '/' will be considered delimiters and a key such as 1/2/3.4 will consist of the entities 1,2,3,4, all matched separately.

Used in J-, I- and C-style CLIs.

The *cli-range-delimiters* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-range-list-syntax

Specifies that elements in a leaf-list or a list should be entered without surrounding brackets and presented as ranges. The element in the list should be separated by a comma. For example:

```
vlan 1,3,10-20,30,32,300-310
```

When this statement is used for lists, the list must have a single key. The elements are be presented as ranges as above.

The type of the list key, or the leaf-list, must be integer based.

Used in J-, I- and C-style CLIs.

The *cli-range-list-syntax* statement can be used in: *leaf-list*, *list*, and *refine*.

tailf:cli-recursive-delete

When generating configuration diffs delete all contents of a container or list before deleting the node.

Applies to C-style

The *cli-recursive-delete* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-remove-before-change

Instructs the CLI engine to generate a no-commnd before modifying an existing instance. It only applies when generating diffs, eg 'show configuration' in C-style.

The *cli-remove-before-change* statement can be used in: *leaf-list*, *list*, *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-replace-all

Specifies that the new leaf-list value(s) should replace the old, as opposed to be added to the old leaf-list.

The *cli-replace-all* statement can be used in: *leaf-list*, *tailf:cli-flat-list-syntax*, and *refine*.

tailf:cli-reset-container

Specifies that all sibling leaves in the container should be reset when this element is set.

When used on a container its content is cleared when set.

The *cli-reset-container* statement can be used in: *leaf*, *list*, *container*, *tailf:symlink*, and *refine*.

tailf:cli-run-template *value*

Specifies a template string to be used by the 'show running-config' command in operational mode. It is primarily intended for displaying config data but non-config data may be included in the template as well.

See the defintion of cli-template-string for more info.

Used in I- and C-style CLIs.

The *cli-run-template* statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-run-template-enter *value*

Specifies a template string to be printed before each list entry is printed.

When used on a container it only has effect when the container also has a `tailf:cli-add-mode`, and when `tailf:cli-show-no` isn't used on the container.

See the definition of `cli-template-string` for more info.

The variable `.reenter` is set to 'true' when the 'show configuration' command is executed and the list or container isn't created. This allow, for example, to display

create foo

when an instance is created

edit foo

when something inside the instance is modified.

Used in I- and C-style CLIs.

The `cli-run-template-enter` statement can be used in: *list*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-run-template-footer value

Specifies a template string to be printed after all list entries are printed.

See the definition of `cli-template-string` for more info.

Used in I- and C-style CLIs.

The `cli-run-template-footer` statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-run-template-legend value

Specifies a template string to be printed before all list entries are printed.

See the definition of `cli-template-string` for more info.

Used in I- and C-style CLIs.

The `cli-run-template-legend` statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-sequence-commands

Specifies that an auto-rendered command should only accept arguments in the same order as they are specified in the YANG model. This, in combination with `tailf:cli-drop-node-name`, can be used to create CLI commands for setting multiple leafs in a container without having to specify the leaf names.

In almost all cases this annotation should be accompanied by the `tailf:cli-compact-syntax` annotation. Otherwise the output from 'show running-config' will not be correct, and the sequence 'save xx' 'load override xx' will not work.

Used in I- and C-style CLIs.

The `cli-sequence-commands` statement can be used in: *list*, *container*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-reset-siblings Specifies that all sibling leaves in the sequence should be reset whenever the first leaf in the sequence is set.

tailf:cli-reset-all-siblings Specifies that all sibling leaves in the container should be reset whenever the first leaf in the sequence is set.

tailf:cli-suppress-warning

tailf:cli-show-config

Specifies that the node will be included when doing a 'show running-configuration', even if it is a non-config node.

Used in I- and C-style CLIs.

The *cli-show-config* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *refine*, and *tailf:symlink*.

tailf:cli-show-long-obu-diffs

Instructs the CLI engine to not generate 'insert' comments when displaying configuration changes of ordered-by user lists, but instead explicitly remove old instances with 'no' and then add the instances following a newly inserted instance. Should not be used together with *tailf:cli-show-obu-comments*

The *cli-show-long-obu-diffs* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-reset-full Indicates that the list should be fully printed out on change.

tailf:cli-show-no

Specifies that an optional leaf node or presence container should be displayed as 'no <name>' when it does not exist. For example, if a leaf 'shutdown' has this property and does not exist, 'no shutdown' is displayed.

Used in I- and C-style CLIs.

The *cli-show-no* statement can be used in: *leaf*, *list*, *leaf-list*, *refine*, *tailf:symlink*, and *container*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-show-obu-comments

Enforces the CLI engine to generate 'insert' comments when displaying configuration changes of ordered-by user lists. Should not be used together with *tailf:cli-show-long-obu-diffs*

The *cli-show-obu-comments* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-show-order-tag *value*

Specifies a custom display order for nodes with the *tailf:cli-show-order-tag* attribute. Nodes will be displayed in the order indicated by a *cli-show-order-taglist* attribute in a parent node.

The scope of a tag reaches until a new taglist is encountered.

Used in I- and C-style CLIs.

The *cli-show-order-tag* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-show-order-taglist *value*

Specifies a custom display order for nodes with the *tailf:cli-show-order-tag* attribute. Nodes will be displayed in the order indicated in the list. Nodes without a tag will be displayed after all nodes with a tag have been displayed.

The scope of a taglist is until a new taglist is encountered.

Used in I- and C-style CLIs.

The *cli-show-order-taglist* statement can be used in: *container*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-show-template *value*

Specifies a template string to be used by the 'show' command in operational mode. It is primarily intended for displaying non-config data but config data may be included in the template as well.

See the definition of *cli-template-string* for more info.

Some restrictions includes not applying templates on a leaf that is the key in a list. It is recommended to use the template directly on the list to format the whole list instead.

Used in J-, I- and C-style CLIs.

The *cli-show-template* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

The following substatements can be used:

tailf:cli-auto-legend Specifies that the legend should be automatically rendered if not already displayed. Useful when using templates for rendering tables.

tailf:cli-show-template-enter *value*

Specifies a template string to be printed before each list entry is printed.

See the definition of *cli-template-string* for more info.

Used in J-, I- and C-style CLIs.

The *cli-show-template-enter* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-show-template-footer *value*

Specifies a template string to be printed after all list entries are printed.

See the definition of *cli-template-string* for more info.

Used in J-, I- and C-style CLIs.

The *cli-show-template-footer* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-show-template-legend *value*

Specifies a template string to be printed before all list entries are printed.

See the definition of `cli-template-string` for more info.

Used in J-, I- and C-style CLIs.

The `cli-show-template-legend` statement can be used in: *list* and *refine*.

tailf:cli-show-with-default

This leaf will be displayed even when it has its default value. Note that this will somewhat result in a slightly different behaviour when you save a config and then load it again. With this setting in place a leaf that has not been configured will be configured after the load.

Used in I- and C-style CLIs.

The `cli-show-with-default` statement can be used in: *leaf*, *refine*, and *tailf:symlink*.

tailf:cli-strict-leafref

Specifies that the leaf should only be allowed to be assigned references to existing instances when the command is executed. Without this annotation the requirement is that the instance exists on commit time.

Used in I- and C-style CLIs.

The `cli-strict-leafref` statement can be used in: *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-key-abbreviation

Key values cannot be abbreviated. The user must always give complete values for keys.

In the J-style CLI this is relevant when using the commands 'delete' and 'edit'.

In the I- and C-style CLIs this is relevant when using the commands 'no', 'show configuration' and for commands to enter submodes.

See also `/confdConfig/cli/allowAbbrevKeys` in `confd.conf(5)`.

The `cli-suppress-key-abbreviation` statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-key-sort

Instructs the CLI engine to not sort the keys in alphabetical order when presenting them to the user during TAB completion.

Used in J-, I- and C-style CLIs.

The `cli-suppress-key-sort` statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-list-no

Specifies that the CLI should not accept deletion of the entire list or leaf-list. Only specific instances should be deletable not the entire list in one command. ie, 'no foo <instance>' should be allowed but not 'no foo'.

Used in I- and C-style CLIs.

The `cli-suppress-list-no` statement can be used in: *leaf-list*, *list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-mode

Instructs the CLI engine to not make a mode of the list node.

Can be used in config nodes only.

Used in I- and C-style CLIs.

The *cli-suppress-mode* statement can be used in: *list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-suppress-no

Specifies that the CLI should not auto-render 'no' commands for this element. An element with this annotation will not appear in the completion list to the 'no' command.

Used in I- and C-style CLIs.

The *cli-suppress-no* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

tailf:cli-suppress-quotes

Specifies that configuration data for a leaf should never be wrapped with quotes. All internal data will be escaped to make sure it can be presented correctly.

Can't be used for keys.

Used in J-, I- and C-style CLIs.

The *cli-suppress-quotes* statement can be used in: *leaf*.

tailf:cli-suppress-range

Means that the integer key should not allow range expressions.

Can be used in key leafs only.

Used in J-, I- and C-style CLIs.

The *cli-suppress-range* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:cli-suppress-warning

tailf:cli-suppress-shortenabled

Suppresses the confd.conf(5) setting /confdConfig/cli/useShortEnabled.

Used in J-, I- and C-style CLIs.

The *cli-suppress-shortenabled* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-show-conf-path

Specifies that the show running-config command cannot be invoked with the path, ie the path is suppressed when auto-rendering show running- config commands for config='true' data.

Used in J-, I- and C-style CLIs.

The *cli-suppress-show-conf-path* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-show-match

Specifies that a specific completion match (i.e., a filter match that appear at list nodes as an alternative to specifying a single instance) to the show command should not be available.

Used in J-, I- and C-style CLIs.

The *cli-suppress-show-match* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

tailf:cli-suppress-show-path

Specifies that the show command cannot be invoked with the path, ie the path is suppressed when auto-rendering show commands for config='false' data.

Used in J-, I- and C-style CLIs.

The *cli-suppress-show-path* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

tailf:cli-suppress-silent-no value

Specifies that the confd.cnof directive cSilentNo should be suppressed for a leaf and that a custom error message should be displayed when the user attempts to delete a non-existing element.

Used in I- and C-style CLIs.

The *cli-suppress-silent-no* statement can be used in: *leaf*, *leaf-list*, *list*, *tailf:symlink*, *container*, and *refine*.

tailf:cli-suppress-table

Instructs the CLI engine to not print the list as a table in the 'show' command.

Can be used in non-config nodes only.

Used in I- and C-style CLIs.

The *cli-suppress-table* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-validation-warning-prompt

Instructs the CLI engine to not prompt the user whether to proceed or not if a warning is generated for this node.

Used in I- and C-style CLIs.

The *cli-suppress-validation-warning-prompt* statement can be used in: *list*, *leaf*, *container*, *leaf-list*, *tailf:symlink*, and *refine*.

tailf:cli-suppress-warning value

Avoid involving specific CLI-extension related YANG statements in warnings related to certain yanger error codes. For a list of yanger error codes do 'yang -e'.

Used in I- and C-style CLIs.

The *cli-suppress-warning* statement can be used in: *tailf:cli-run-template-enter*, *tailf:cli-sequence-commands*, *tailf:cli-hide-in-submode*, *tailf:cli-boolean-no*, *tailf:cli-compact-syntax*, *tailf:cli-break-sequence-commands*, *tailf:cli-show-long-obu-diffs*, *tailf:cli-show-obu-comments*, *tailf:cli-suppress-range*,

tailf:cli-suppress-mode, *tailf:cli-custom-range*, *tailf:cli-custom-range-actionpoint*, *tailf:cli-custom-range-enumerator*, *tailf:cli-drop-node-name*, *tailf:cli-add-mode*, *tailf:cli-mode-name*, *tailf:cli-incomplete-command*, *tailf:cli-full-command*, *tailf:cli-mode-name-actionpoint*, *tailf:cli-optional-in-sequence*, *tailf:cli-prefix-key*, *tailf:cli-show-no*, *tailf:cli-show-order-tag*, *tailf:cli-diff-dependency*, and *container*.

tailf:cli-suppress-wildcard

Means that the list does not allow wildcard expressions in the 'show' pattern.

See also `/confdConfig/cli/allowWildcard` in `confd.conf(5)`.

Used in J-, I- and C-style CLIs.

The *cli-suppress-wildcard* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-table-footer value

Specifies a template string to be printed after all list entries are printed.

Used in J-, I- and C-style CLIs.

The *cli-table-footer* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-table-legend value

Specifies a template string to be printed before all list entries are printed.

Used in J-, I- and C-style CLIs.

The *cli-table-legend* statement can be used in: *list*, *tailf:symlink*, and *refine*.

tailf:cli-trim-default

Do not display value if it is same as default.

Used in I- and C-style CLIs.

The *cli-trim-default* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

tailf:cli-value-display-template value

Specifies a template string to be used when forming the value of a leaf for display. Note that other leaves cannot be referenced from a display template of one leaf. The only value accessible is the leaf's own value, accessed through `$(.)`.

See the definition of `cli-template-string` for more info.

Used in J-, I- and C-style CLIs.

The *cli-value-display-template* statement can be used in: *leaf*, *tailf:symlink*, and *refine*.

YANG TYPES

cli-template-string

A template is a text string which is expanded by the CLI engine, and then displayed to the user.

The template may contain a mix of text and expandable entries. Expandable entries all start with `$(` and end with a matching `)`. Parentheses and dollar signs need to be quoted in plain text.

(Disclaimer: tailf:cli-template-string will not respect all CLI YANG extensions existing from expandable entries. For instance, tailf:cli-no-name-on-delete will have no effect when the value of a node with this extension is fetched as a result of expanding CLI templates.)

The template is expanded as follows:

A parameter is either a relative or absolute path to a leaf element (eg /foo/bar, foo/bar), or one of the builtin variables: .selected, .entered, .legend_shown, .user, .groups, .ip, .display_groups, .path, .ipath or .licounter. In addition the variables .spath and .ispath are available when a command is executed from a show path.

.selected

The .selected variable contains the list of selected paths to be shown. The show template can inspect this element to determine if a given element should be displayed or not. For example:

```
$(.selected~=hwaddr?HW Address)
```

.entered

The .entered variable is true if the "entered" text has been displayed (either the auto generated text or a showTemplateEnter). This is useful when having a non-table template where each instance should have a text.

```
$(.entered?:host $(name))
```

.legend_shown

The .legend_shown variable is true if the "legend" text has been displayed (either the auto generated table header or a showTemplateLegend). This is useful to inspect when displaying a table row. If the user enters the path to a specific instance the builtin table header will not be displayed and the showTemplateLegend will not be invoked and it may be useful to render the legend specifically for this instance.

```
$(.legend_shown!=true?Address Interface)
```

.user

The .user variable contains the name of the current user. This can be used for differentiating the content displayed for a specific user, or in paths. For example:

```
$(user{$(.user)}/settings)
```

.groups

The .groups variable contains the a list of groups that the user belongs to.

.display_groups

The .display_groups variable contains a list of selected display groups. This can be used to display different content depending on the selected display group. For example:

```
$(.display_groups~=details?details...)
```

.ip

The .ip variable contains the ip address that the user connected from.

.path

The .path variable contains the path to the entry, formatted in CLI style.

`.ipath`

The `.ipath` variable contains the path to the entry, formatted in template style.

`.spath`

The `.spath` variable contains the show path, formatted in CLI style.

`.ispath`

The `.ispath` variable contains the show path, formatted in template style.

`.licounter`

The `.licounter` variable contains a counter that is incremented for each instance in a list. This means that it will be 0 in the legend, contain the total number of list instances in the footer and something in between in the basic show template.

`$(parameter)`

The value of 'parameter' is substituted.

`$(cond?word1:word2)`

The expansion of 'word1' is substituted if 'cond' evaluates to true, otherwise the expansion of 'word2' is substituted.

'cond' may be one of

`parameter`

Evaluates to true if the node exists.

`parameter == <value>`

Evaluates to true if the value of the parameter equals <value>.

`parameter != <value>`

Evaluates to true if the value of the parameter does not equal <value>

`parameter ~= <value>`

Provided that the value of the parameter is a list (i.e., the node that the parameter refers to is a leaf-list), this expression evaluates to true if <value> is a member of the list.

Note that it is also possible to omit 'word2' in order to print the entire statement, or nothing. As an example `$(conf?word1)` will print 'word1' if conf exists, otherwise it will print nothing.

`$(cond??word1)`

Double question marks can be used to achieve the same effect as above, but with the distinction that the 'cond' variable needs to be explicitly configured, in order to be evaluated as existing. This is needed in the case of evaluating leafs with default values, where the single question mark operator would evaluate to existing even if not explicitly configured.

`$(parameter|filter)`

The value of 'parameter' processed by 'filter' is substituted. Filters may be either one of the built-ins or a customized filter defined in a callback. See `/confdConfig/cli/templateFilter`.

A built-in 'filter' may be one of:

capfirst

Capitalizes the first character of the value.

lower

Converts the value into lowercase.

upper

Converts the value into uppercase.

filesizeformat

Formats the value in a human-readable format (e.g., '13 KB', '4.10 MB', '102 bytes' etc), where K means 1024, M means 1024*1024 etc.

When used without argument the default number of decimals displayed is 2. When used with a numeric integer argument, filesizeformat will display the given number of decimal places.

humanreadable

Similar to filesizeformat except no bytes suffix is added (e.g., '13.00 k', '4.10 M' '102' etc), where k means 1000, M means 1000*1000 etc.

When used without argument the default number of decimals displayed is 2. When used with a numeric integer argument, humanreadable will display the given number of decimal places.

commasep

Separate the numerical values into groups of three digits using a comma (e.g., 1234567 -> 1,234,567)

hex

Display integer as hex number. An argument can be used to indicate how many digits should be used in the output. If the hex number is too long it will be truncated at the front, if it is too short it will be padded with zeros at the front. If the width is a negative number then at most that number of digits will be used, but short numbers will not be padded with zeroes. Another argument can be given to indicate if the hex numbers should be written with lower or upper case.

For example:

value	Template	Output
12345	{{ value hex }}	3039
12345	{{ value hex:2 }}	39
12345	{{ value hex:8 }}	00003039
12345	{{ value hex:-8 }}	3039
14911	{{ value hex:-8:upper }}	3A3F
14911	{{ value hex:-8:lower }}	3a3f

hexlist

Display integer as hex number with : between pairs. An argument can be used to indicate how many digits should be used in the output. If the hex number is too long it will be truncated at the front, if it is too short it will be padded with zeros at the front. If the width is a negative number then at most that number of digits will be used, but short numbers will not be padded with zeroes. Another argument can be given to indicate if the hex numbers should be written with lower or upper case.

For example:

value	Template	Output
12345	{{ value hexlist }}	30:39
12345	{{ value hexlist:2 }}	39
12345	{{ value hexlist:8 }}	00:00:30:39
12345	{{ value hexlist:-8 }}	30:39
14911	{{ value hexlist:-8:upper }}	3A:3F
14911	{{ value hexlist:-8:lower }}	3a:3f

floatformat

Used for type 'float' in tailf-xsd-types. We recommend that the YANG built-in type 'decimal64' is used instead of 'float'.

When used without an argument, rounds a floating-point number to one decimal place -- but only if there is a decimal part to be displayed.

For example:

value	Template	Output
34.23234	{{ value floatformat }}	34.2
34.00000	{{ value floatformat }}	34
34.26000	{{ value floatformat }}	34.3

If used with a numeric integer argument, floatformat rounds a number to that many decimal places. For example:

value	Template	Output
34.23234	{{ value floatformat:3 }}	34.232
34.00000	{{ value floatformat:3 }}	34.000
34.26000	{{ value floatformat:3 }}	34.260

If the argument passed to floatformat is negative, it will round a number to that many decimal places -- but only if there's a decimal part to be displayed. For example:

value	Template	Output
34.23234	{{ value floatformat:-3 }}	34.232
34.00000	{{ value floatformat:-3 }}	34
34.26000	{{ value floatformat:-3 }}	34.260

Using floatformat with no argument is equivalent to using floatformat with an argument of -1.

ljust:width

Left-align the value given a width.

rjust:width

Right-align the value given a width.

trunc:width

Truncate value to a given width.

lower

Convert the value into lowercase.

upper

Convert the value into uppercase.

show:<dictionary>

Substitutes the result of invoking the default display function for the parameter. The dictionary can be used for introducing own variables that can be accessed in the same manner as builtin variables. The user defined variables overrides builtin variables. The dictionary is specified as a string on the following form:

(key=value):(key=value)*

For example, with the following expression:

\$(foo|show:myvar1=true:myvar2=Interface)

the user defined variables can be accessed like this:

\$(.myvar1!=true?Address) \$(.myvar2)

A special case is the dict variable 'indent'. It controls the indentation level of the displayed path. The current indent level can be incremented and decremented using += and -=.

For example:

\$(foobar|show:indent=+2) \$(foobar|show:indent=-1) \$(foobar|show:indent=10)

Another special case is the dict variable 'noalign'. It may be used to suppress the default aligning that may occur when displaying an element.

For example:

\$(foobar|show:noalign)

dict:<dictionary>

Translates the value using the dictionary. Can for example be used for displaying on/off instead of true/false. The dictionary is specified as a string on the following form:

(key=value):(key=value)*

For example, with the following expression:

\$(foo|dict:true=on:false=off)

if the leaf 'foo' has value 'true', it is displayed as 'on', and if its value is 'false' it is displayed as 'off'.

Nested invocations are allowed, ie it is possible to have expressions like `$(state|dict:yes=Yes:no=No)|rjust:14)`, or `$(/foo{$(../bar)})`

For example:

```
list interface {
  key name;
  leaf name { ... }
  leaf status { ... }
  container line {
    leaf status { ... }
  }
  leaf mtu { ... }
  leaf bw { ... }
  leaf encapsulation { ... }
  leaf loopback { ... }
  tailf:cli-show-template
    '$(name) is administratively $(status),'
  + ' line protocol is $(line/status)\n'
  + 'MTU $(mtu) bytes, BW $(bw|humanreadable)bit, \n'
  + 'Encap $(encapsulation|upper), $(loopback?:loopback not set)\n';
```

}

SEE ALSO

The User Guide

`ncsc(1)`

`tailf_yang_extensions(5)`

NCS Yang compiler

Tail-f YANG extensions



Name

tailf_yang_extensions — Tail-f YANG extensions

Synopsis

tailf:action
tailf:actionpoint
tailf:alt-name
tailf:annotate
tailf:annotate-module
tailf:callpoint
tailf:cdb-oper
tailf:code-name
tailf:confirm-text
tailf:default-ref
tailf:dependency
tailf:display-column-name
tailf:display-groups
tailf:display-hint
tailf:display-status-name
tailf:display-when
tailf:error-info
tailf:exec
tailf:export
tailf:hidden
tailf:id
tailf:id-value
tailf:indexed-view
tailf:info
tailf:info-html
tailf:java-class-name
tailf:junos-val-as-xml-tag
tailf:junos-val-with-prev-xml-tag
tailf:key-default

tailf:link
tailf:lower-case
tailf:meta-data
tailf:ncs-device-type
tailf:ned-data
tailf:ned-default-handling
tailf:ned-ignore-compare-config
tailf:no-dependency
tailf:no-leafref-check
tailf:non-strict-leafref
tailf:operation
tailf:override-auto-dependencies
tailf:path-filters
tailf:secondary-index
tailf:snmp-delete-value
tailf:snmp-exclude-object
tailf:snmp-lax-type-check
tailf:snmp-mib-module-name
tailf:snmp-name
tailf:snmp-ned-accessible-column
tailf:snmp-ned-delete-before-create
tailf:snmp-ned-modification-dependent
tailf:snmp-ned-recreate-when-modified
tailf:snmp-ned-set-before-row-modification
tailf:snmp-oid
tailf:snmp-row-status-column
tailf:sort-order
tailf:sort-priority
tailf:step
tailf:structure
tailf:suppress-echo
tailf:symlink
tailf:transaction

```
tailf:typepoint
tailf:unique-selector
tailf:validate
tailf:value-length
tailf:writable
tailf:xpath-root
```

DESCRIPTION

This manpage describes all the Tail-f extensions to YANG. The YANG extensions consist of YANG statements and XPath functions to be used in YANG data models.

The YANG source file `$NCS_DIR/src/ncs/yang/tailf-common.yang` gives the exact YANG syntax for all Tail-f YANG extension statements - using the YANG language itself.

Most of the concepts implemented by the extensions listed below are described in the NSO User Guide. For example user defined validation is described in the Validation chapter. The YANG syntax is described here though.

YANG STATEMENTS

tailf:action name

Defines an action (method) in the data model.

When the action is invoked, the instance on which the action is invoked is explicitly identified by an hierarchy of configuration or state data.

The action statement can have either a 'tailf:actionpoint' or a 'tailf:exec' substatement. If the action is implemented as a callback in an application daemon, 'tailf:actionpoint' is used, whereas 'tailf:exec' is used for an action implemented as a standalone executable (program or script). Additionally, 'action' can have the same substatements as the standard YANG 'rpc' statement, e.g., 'description', 'input', and 'output'.

For example:

```
container sys {
  list interface {
    key name;
    leaf name {
      type string;
    }
    tailf:action reset {
      tailf:actionpoint my-ap;
      input {
        leaf after-seconds {
          mandatory false;
          type int32;
        }
      }
    }
  }
}
```

We can also add a 'tailf:confirm-text', which defines a string to be used in the user interfaces to prompt the user for confirmation before the action is executed. The optional 'tailf:confirm-default' and 'tailf:cli-batch-

`confirm-default` can be set to control if the default is to proceed or to abort. The latter will only be used during batch processing in the CLI (e.g. non-interactive mode).

```
tailf:action reset {
  tailf:actionpoint my-ap;
  input {
    leaf after-seconds {
      mandatory false;
      type int32;
    }
  }
  tailf:confirm-text 'Really want to do this?' {
    tailf:confirm-default true;
  }
}
```

The `tailf:actionpoint` statement can have a `tailf:opaque` substatement, to define an opaque string that is passed to the callback function.

```
tailf:action reset {
  tailf:actionpoint my-ap {
    tailf:opaque 'reset-interface';
  }
  input {
    leaf after-seconds {
      mandatory false;
      type int32;
    }
  }
}
```

When we use the `tailf:exec` substatement, the argument to `exec` specifies the program or script that should be executed. For example:

```
tailf:action reboot {
  tailf:exec '/opt/sys/reboot.sh' {
    tailf:args '-c $(context) -p $(path)';
  }
  input {
    leaf when {
      type enumeration {
        enum now;
        enum 10secs;
        enum 1min;
      }
    }
  }
}
```

The *action* statement can be used in: *augment*, *list*, *container*, and *grouping*.

The following substatements can be used:

tailf:actionpoint

tailf:alt-name

tailf:cli-mount-point

tailf:cli-configure-mode

tailf:cli-operational-mode

tailf:cli-oper-info

tailf:code-name

tailf:confirm-text

tailf:display-when

tailf:exec

tailf:hidden

tailf:info

tailf:info-html

tailf:actionpoint *name*

Identifies the callback in a data provider that implements the action. See `confd_lib_dp(3)` for details on the API.

The *actionpoint* statement can be used in: *rpc*, *action*, *tailf:action*, and *refine*.

The following substatements can be used:

tailf:opaque Defines an opaque string which is passed to the callback function in the context.

tailf:internal For internal ConfD / NCS use only.

tailf:alt-name *name*

This property is used to specify an alternative name for the node in the CLI. It is used instead of the node name in the CLI, both for input and output.

The *alt-name* statement can be used in: *rpc*, *action*, *leaf*, *leaf-list*, *list*, *container*, and *refine*.

tailf:annotate *target*

Annotates an existing statement with a 'tailf' statement or a validation statement. This is useful in order to add tailf statements to a module without touching the module source. Annotation statements can be put in a separate annotation module, and then passed to 'confdc' (or 'pyang') when the original module is compiled.

Any 'tailf' statement, except 'symlink' and 'action' can be annotated. The statements 'symlink' and 'action' modifies the data model, and are thus not allowed.

The validation statements 'must', 'min-elements', 'max-elements', 'mandatory', 'unique', and 'when' can also be annotated.

A 'description' can also be annotated.

'tailf:annotate' can occur on the top-level in a module, or in another 'tailf:annotate' statement.

The argument is a 'schema-nodeid', i.e. the same as for 'augment', or a '*'. It identifies a target node in the schema tree to annotate with new statements. The special value '*' can be used within another 'tailf:annotate' statement, to select all children for annotation.

The target node is searched for after 'uses' and 'augment' expansion. All substatements to 'tailf:annotate' are treated as if they were written inline in the target node, with the exception of any 'tailf:annotate' substatements. These are treated recursively. For example, the following snippet adds one callpoint to /x and one to /x/y:

```

tailf:annotate /x {
    tailf:callpoint xcp;
    tailf:annotate y {
        tailf:callpoint ycp;
    }
}

```

The *annotate* statement can be used in: *module* and *submodule*.

The following substatements can be used:

tailf:annotate

tailf:annotate-module *module-name*

Annotates an existing module or submodule statement with a 'tailf' statement. This is useful in order to add tailf statements to a module without touching the module source. Annotation statements can be put in a separate annotation module, and then passed to 'confdc' (or 'pyang') when the original module is compiled.

'tailf:annotate-module' can occur on the top-level in a module, and is used to add 'tailf' statements to the module statement itself.

The argument is a name of the module or submodule to annotate.

The *annotate-module* statement can be used in: *module*.

The following substatements can be used:

tailf:snmp-oid

tailf:snmp-mib-module-name

tailf:id

tailf:id-value

tailf:export

tailf:unique-selector

tailf:annotate-statement Annotates an existing statement with a 'tailf' statement, a validation statement, or a type restriction statement. This is useful in order to add tailf statements to a module without touching the module source. Annotation statements can be put in a separate annotation module, and then passed to 'confdc' (or 'pyang') when the original module is compiled.

Any 'tailf' statement, except 'symlink' and 'action' can be annotated. The statements 'symlink' and 'action' modifies the data model, and are thus not allowed.

The validation statements 'must', 'min-elements', 'max-elements', 'mandatory', 'unique', and 'when' can also be annotated.

The type restriction statement 'pattern' can also be annotated.

A 'description' can also be annotated.

The argument is an XPath-like expression that selects a statement to annotate. The syntax is:

```
<statement-name> ( '[' <arg-name> '=' <arg-value> ']' )
```

where <statement-name> is the name of the statement to annotate, and if there are more than one such statement in the parent, <arg-value> is the quoted value of the statement's argument.

All substatements to 'tailf:annotate-statement' are treated as if they were written inline in the target node, with the exception of any 'tailf:annotate-statement' substatements. These are treated recursively.

For example, given the grouping:

```
grouping foo { leaf bar { type string; } leaf baz { type string; } }
```

the following snippet adds a callpoint to the leaf 'baz':

```
tailf:annotate-statement grouping[name='foo'] { tailf:annotate-statement leaf[name='baz'] { tailf:callpoint xcp; } }
```

tailf:callpoint *id*

Identifies a callback in a data provider. A data provider implements access to external data, either configuration data in a database or operational data. By default ConfD uses the embedded database (CDB) to store all data. However, some or all of the configuration data may be stored in an external source. In order for ConfD to be able to manipulate external data, a data provider registers itself using the callpoint id as described in `confd_lib_dp(3)`.

A callpoint is inherited to all child nodes unless another 'callpoint' or an 'cdb-oper' is defined.

The *callpoint* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *refine*, and *grouping*.

The following substatements can be used:

tailf:config If this statement is present, the callpoint is applied to nodes with a matching value of their 'config' property.

tailf:transform If set to 'true', the callpoint is a transformation callpoint. How transformation callpoints are used is described in the 'Transformations, Hooks, Hidden Data and Symlinks' chapter in the User's Guide.

tailf:set-hook Set hooks are a means to associate user code to the transaction. Whenever an element gets written, created, or deleted, user code gets invoked and can optionally write more data into the same transaction.

The difference between set- and transaction hooks are that set hooks are invoked immediately when a write operation is requested by a north bound agent, and transaction hooks are invoked at commit time.

The value 'subtree' means that all nodes in the configuration below where the hook is defined are affected.

The value 'object' means that the hook only applies to the list where it is defined, i.e. it applies to all child nodes that are not themselves lists.

The value 'node' means that the hook only applies to the node where it is defined and none of its children.

For more details on hooks, see the 'Transformations, Hooks, Hidden Data and Symlinks' chapter in the User's Guide.

tailf:transaction-hook Transaction hooks are a means to associate user code to the transaction. Whenever an element gets written, created, or deleted, user code gets invoked and can optionally write more data into the same transaction.

The difference between set- and transaction hooks are that set hooks are invoked immediately when an element is modified, but transaction hooks are invoked at commit time.

The value 'subtree' means that all nodes in the configuration below where the hook is defined are affected.

The value 'object' means that the hook only applies to the list where it is defined, i.e. it applies to all child nodes that are not themselves lists.

The value 'node' means that the hook only applies to the node where it is defined and none of its children.

For more details on hooks, see the 'Transformations, Hooks, Hidden Data and Symlinks' chapter in the User's Guide.

tailf:cache If set to 'true', the operational data served by the callpoint will be cached by ConfD. If set to 'true' in a node that represents configuration data, the statement 'tailf:config' must be present and set to 'false'. This feature is further described in the section 'Caching operational data' in the 'Operational data' chapter in the User's Guide.

tailf:opaque Defines an opaque string which is passed to the callback function in the context.

tailf:internal For internal ConfD / NCS use only.

tailf:cdb-oper

Indicates that operational data nodes below this node are stored in CDB.

The *cdb-oper* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:persistent If it is set to 'true', the operational data is stored on disk. If set to 'false', the operational data is not persistent across ConfD restarts. The default is 'false'.

tailf:code-name *name*

Used to give another name to the enum or node name in generated header files. This statement is typically used to avoid name conflicts if there is a data node with the same name as the enumeration, if there are multiple enumerations in different types with the same name but different values, or if there are multiple node names that are mapped to the same name in the header file.

The *code-name* statement can be used in: *enum*, *bit*, *leaf*, *leaf-list*, *list*, *container*, *rpc*, *action*, *identity*, *notification*, and *tailf:action*.

tailf:confirm-text *text*

A string which is used in the user interfaces to prompt the user for confirmation before the action is executed. The optional 'confirm-default' and 'cli-batch-confirm-default' can be set to control if the default is to proceed or to abort. The latter will only be used during batch processing in the CLI (e.g. non-interactive mode).

The *confirm-text* statement can be used in: *rpc*, *action*, and *tailf:action*.

The following substatements can be used:

tailf:confirm-default Specifies if the default is to proceed or abort the action when a confirm-text is set. If this value is not specified, a ConfD global default value can be set in `clispec(5)`.

tailf:cli-batch-confirm-default

tailf:default-ref *path*

This statement defines a dynamic default value. It is a reference to some other leaf in the datamodel. If no value has been set for this leaf, it defaults to the value of the leaf that the 'default-ref' argument points to.

The textual format of a 'default-ref' is an XPath location path with no predicates.

The type of the leaf with a 'default-ref' will be set to the type of the referred leaf. This means that the type statement in the leaf with the 'default-ref' is ignored, but it SHOULD match the type of the referred leaf.

Here is an example, where a group without a 'hold-time' will get as default the value of another leaf up in the hierarchy:

```
leaf hold-time {
    mandatory true;
    type int32;
}
list group {
    key 'name';
    leaf name {
        type string;
    }
    leaf hold-time {
        type int32;
        tailf:default-ref '../..hold-time';
    }
}
```

The *default-ref* statement can be used in: *leaf* and *refine*.

tailf:dependency *path*

This statement is used to specify that the must or when expression or validation function depends on a set of subtrees in the data store. Whenever a node in one of those subtrees are modified, the must or when expression is evaluated, or validation code executed.

The textual format of a 'dependency' is an XPath location path with no predicates.

If the node that declares the dependency is a leaf, there is an implicit dependency to the leaf itself.

For example, with the leafs below, the validation code for 'vp' will be called whenever 'a' or 'b' is modified.

```
leaf a {
    type int32;
    tailf:validate vp {
        tailf:dependency '../b';
    }
}
leaf b {
    type int32;
}
```

For 'when' and 'must' expressions, the compiler can derive the dependencies automatically from the XPath expression in most cases. The exception is if any wildcards are used in the expression.

For 'when' expressions to work, a 'tailf:dependency' statement must be given, unless the compiler can figure out the dependency by itself.

Note that having 'must' expressions or a 'tailf:validate' statement without dependencies impacts the overall performance of the system, since all such 'must' expressions or validation functions are evaluated at every commit.

The *dependency* statement can be used in: *must*, *when*, and *tailf:validate*.

The following substatements can be used:

tailf:xpath-root

tailf:display-column-name *name*

This property is used to specify an alternative column name for the leaf in the CLI. It is used when displaying the leaf in a table in the CLI.

The *display-column-name* statement can be used in: *leaf*, *leaf-list*, and *refine*.

tailf:display-groups *value*

This property is used in the CLI when 'enableDisplayGroups' has been set to true in the confd.conf(5) file. Display groups are used to control which elements should be displayed by the show command.

The argument is a space-separated string of tags.

In the J-style CLI the 'show status', 'show table' and 'show all' commands use display groups. In the C- and I-style CLIs the 'show <pattern>' command uses display groups.

If no display groups are specified when running the commands, the node will be displayed if it does not have the 'display-groups' property, or if the property value includes the special value 'none'.

If display groups are specified when running the command, then the node will be displayed only if its 'display-group' property contains one of the specified display groups.

The *display-groups* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

tailf:display-hint *hint*

This statement can be used to add a display-hint to a leaf or typedef of type binary. The display-hint is used in the CLI and WebUI instead of displaying the binary as a base64-encoded string. It is also used for input.

The value of a 'display-hint' is defined in RFC 2579.

For example, with the display-hint value '1x:', the value is printed and inputted as a colon-separated hex list.

The *display-hint* statement can be used in: *leaf* and *typedef*.

tailf:display-status-name *name*

This property is used to specify an alternative name for the element in the CLI. It is used when displaying status information in the C- and I-style CLIs.

The *display-status-name* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

tailf:display-when *condition*

The argument contains an XPath expression which specifies when the node should be displayed in the CLI and WebUI. For example, when the CLI performs completion, and one of the candidates is a node with a 'display-when' expression, the expression is evaluated by the CLI. If the XPath expression evaluates to true, the node is shown as a possible completion candidate, otherwise not.

For a list, the display-when expression is evaluated once for the entire list. In this case, the XPath context node is the list's parent node.

This feature is further described in the 'Transformations, Hooks, Hidden Data and Symlinks' chapter in the User Guide.

The *display-when* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

The following substatements can be used:

tailf:xpath-root

tailf:error-info

Declares a set of data nodes to be used in the NETCONF <error-info> element.

A data provider can use one of the `confd_*_seterr_extended_info()` functions (see `confd_lib_dp(3)`) to set these data nodes on errors.

This statement may be used multiple times.

For example:

```
tailf:error-info {
  leaf severity {
    type enumeration {
      enum info;
      enum error;
      enum critical;
    }
  }
  container detail {
    leaf class {
      type uint8;
    }
    leaf code {
      type uint8;
    }
  }
}
```

The *error-info* statement can be used in: *module* and *submodule*.

tailf:exec cmd

Specifies that the rpc or action is implemented as an OS executable. The argument 'cmd' is the path to the executable file. If the command is in the \$PATH of ConfD, the 'cmd' can be just the name of the executable.

The *exec* statement can be used in: *rpc*, *action*, and *tailf:action*.

The following substatements can be used:

tailf:args Specifies arguments to send to the executable when it is invoked by ConfD. The argument 'value' is a space separated list of argument strings. It may contain variables on the form \$(variablename). These variables will be expanded before the command is executed. The following variables are always available:

\$(user) The name of the user which runs the operation.

\$(groups) A comma separated string of the names of the groups the user belongs to.

\$(ip) The source ip address of the user session.

\$(uid) The user id of the user.

\$(gid) The group id of the user.

When the parent 'exec' statement is a substatement of 'action', the following additional variablenames are available:

\$(keypath) The path that identifies the parent container of 'action' in string keypath form, e.g., '/sys:host{earth}/interface{eth0}'.

\$(path) The path that identifies the parent container of 'action' in CLI path form, e.g., 'host earth interface eth0'.

\$(context) cli | webui | netconf | any string provided by MAAPI

For example: args '-user \$(user) \$(uid)'; might expand to: -user bob 500

tailf:uid Specifies which user id to use when executing the command.

If 'uid' is an integer value, the command is run as the user with this user id.

If 'uid' is set to either 'user', 'root' or an integer user id, the ConfD daemon must have been started as root (or setuid), or the ConfD executable program 'cmdwrapper' must have setuid root permissions.

tailf:gid Specifies which group id to use when executing the command.

If 'gid' is an integer value, the command is run as the group with this group id.

If 'gid' is set to either 'user', 'root' or an integer group id, the ConfD daemon must have been started as root (or setuid), or the ConfD executable program 'cmdwrapper' must have setuid root permissions.

tailf:wd Specifies which working directory to use when executing the command. If not given the command is executed from the homedir of the user logged in to ConfD.

tailf:global-no-duplicate Specifies that only one instance with the same name can be run at any one time in the system. The command can be started either from the CLI, the WebUI or through NETCONF. If a client tries to execute this command while another operation with the same 'global-no-duplicate' name is running, a 'resource-denied' error is generated.

tailf:raw-xml Specifies that ConfD should not convert the RPC XML parameters to command line arguments. Instead, ConfD just passes the raw XML on stdin to the program.

This statement is not allowed in 'tailf:action'.

tailf:interruptible Specifies whether the client can abort the execution of the executable.

tailf:interrupt This statement specifies which signal is sent to executable by ConfD in case the client terminates or aborts the execution.

If not specified, 'sigkill' is sent.

tailf:export agent

Makes this data model visible in the northbound interface 'agent'.

This statement makes it possible to have a data model visible through some northbound interface but not others. For example, if a MIB is used to generate a YANG module, the resulting YANG module can be exposed through SNMP only.

Use the special agent 'none' to make the data model completely hidden to all northbound interfaces.

The agent can also be a free-form string. In this case, the data model will be visible to maapi applications using this string as its 'context'.

The *export* statement can be used in: *module*.

tailf:hidden tag

This statement can be used to hide a node from some, or all, northbound interfaces. All nodes with the same value are considered a hide group and are treated the same with regards to being visible or not in a northbound interface.

A node with an hidden property is not shown in the northbound user interfaces (CLI and Web UI) unless an 'unhide' operation has been performed in the user interface.

The hidden value 'full' indicates that the node should be hidden from all northbound interfaces, including programmatical interfaces such as NETCONF.

The value '*' is not valid.

A hide group can be unhidden only if this has been explicitly allowed in the `confd.conf(5)` daemon configuration.

Multiple hide groups can be specified by giving this statement multiple times. The node is shown if any of the specified hide groups has been given in the 'unhide' operation.

Note that if a mandatory node is hidden, a hook callback function (or similar) might be needed in order to set the element.

The *hidden* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:action*, *refine*, *tailf:symlink*, *rpc*, and *action*.

tailf:id name

This statement is used when old confspec models are translated to YANG. It needs to be present if systems deployed with data based on confspecs are updated to YANG based data models.

In confspec, the 'id' of a data model was a string that never would change, even if the namespace URI would change. It is not needed in YANG, since the namespace URI cannot change as a module is updated.

This statement is typically present in YANG modules generated by `cs2yang`. If no live upgrade needs to be done from a confspec based system to a YANG based system, this statement can be removed from such a generated module.

The *id* statement can be used in: *module*.

tailf:id-value value

This statement lets you specify a hard wired numerical id value to associate with the parent node. This id value is normally auto generated by `confdc` and is used when working with the ConfD API to refer to a tag name, to avoid expensive string comparison. Under certain rare circumstances this auto generated hash value may collide with a hash value generated for a node in another data model. Whenever such a collision occurs the ConfD daemon fails to start and instructs the developer to use the 'id-value' statement to resolve the collision.

A thorough discussion on id-value can be found in the section Hash Values and the id-value Statement in the YANG chapter in the User Guide.

The *id-value* statement can be used in: *module*, *leaf*, *leaf-list*, *list*, *container*, *rpc*, *action*, *identity*, *notification*, *choice*, *case*, and *tailf:action*.

tailf:indexed-view

This element can only be used if the list has a single key of an integer type.

It is used to signal that lists instances uses an indexed view, i.e., making it possible to insert a new list entry at a certain position. If a list entry is inserted at a certain position, list entries following this position are automatically renumbered by the system, if needed, to make room for the new entry.

This statement is mainly provided for backwards compatibility with confspecs. New data models should consider using YANG's ordered-by user statement instead.

The *indexed-view* statement can be used in: *list*.

The following substatements can be used:

tailf:auto-compact If an indexed-view list is marked with this statement, it means that the server will automatically renumber entries after a delete operation so that the list entries are strictly monotonically increasing, starting from 1, with no holes. New list entries can either be inserted anywhere in the list, or created at the end; but it is an error to try to create a list entry with a key that would result in a hole in the sequence.

For example, if the list has entries 1,2,3 it is an error to create entry 5, but correct to create 4.

tailf:info text

Contains a textual description of the definition, suitable for being presented to the CLI and WebUI users.

The first sentence of this textual description is used in the CLI as a summary, and displayed to the user when a short explanation is presented.

The 'description' statement is related, but targeted to the module reader, rather than the CLI or WebUI user.

The info string may contain a ';' keyword. It is used in type descriptions for leafs when the builtin type info needs to be customized. A 'normal' info string describing a type is assumed to contain a short textual description. When ';' is present it works as a delimiter where the text before the keyword is assumed to contain a short description and the text after the keyword a long(er) description. In the context of completion in the CLI the text will be nicely presented in two columns where both descriptions are aligned when displayed.

The *info* statement can be used in: *typedef*, *leaf*, *leaf-list*, *list*, *container*, *rpc*, *action*, *identity*, *type*, *enum*, *bit*, *length*, *pattern*, *range*, *refine*, *action*, *tailf:action*, *tailf:symlink*, and *tailf:cli-exit-command*.

tailf:info-html text

This statement works exactly as 'tailf:info', with the exception that it can contain HTML markup. The WebUI will display the string with the HTML markup, but the CLI will remove all HTML markup before displaying the string to the user. In most cases, using this statement avoids using special descriptions in webspecs and clispecs.

If this statement is present, 'tailf:info' cannot be given at the same time.

The *info-html* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *rpc*, *action*, *identity*, *tailf:action*, *tailf:symlink*, and *refine*.

tailf:java-class-name name

Used to give another name than the default name to generated Java classes. This statement is typically used to avoid name conflicts in the Java classes.

The *java-class-name* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

tailf:junos-val-as-xml-tag

Internal extension to handle non-YANG JUNOS data models. Use only for key enumeration leafs.

The *junos-val-as-xml-tag* statement can be used in: *leaf*.

tailf:junos-val-with-prev-xml-tag

Internal extension to handle non-YANG JUNOS data models. Use only for keys where previous key is marked with 'tailf:junos-val-as-xml-tag'.

The *junos-val-with-prev-xml-tag* statement can be used in: *leaf*.

tailf:key-default *value*

Must be used for key leaves only.

Specifies a value that the CLI and WebUI will use when a list entry is created, and this key leaf is not given a value.

If one key leaf has a key-default value, all key leaves that follow this key leaf must also have key-default values.

The *key-default* statement can be used in: *leaf*.

tailf:link *target*

This statement specifies that the data node should be implemented as a link to another data node, called the target data node. This means that whenever the node is modified, the system modifies the target data node instead, and whenever the data node is read, the system returns the value of target data node.

Note that if the data node is a leaf, the target node **MUST** also be a leaf, and if the data node is a leaf-list, the target node **MUST** also be a leaf-list.

Note that the type of the data node **MUST** be the same as the target data node. Currently the compiler cannot check this.

The argument is an XPath absolute location path. If the target lies within lists, all keys must be specified. A key either has a value, or is a reference to a key in the path of the source node, using the function `current()` as starting point for an XPath location path. For example:

```
/a/b[k1='paul'][k2=current()../k]/c
```

The *link* statement can be used in: *leaf* and *leaf-list*.

The following substatements can be used:

tailf:inherit-set-hook This statement specifies that a 'tailf:set-hook' statement should survive through symlinks. If set to true a set hook gets called as soon as the value is set via a symlink but also during commit. The normal behaviour is to only call the set hook during commit time.

tailf:lower-case

Use for config false leaves and leaf-lists only.

This extension serves as a hint to the system that the leaf's type has the implicit pattern '[^A-Z]*', i.e., all strings returned by the data provider are lower case (in the 7-bit ASCII range).

The CLI uses this hint when it is run in case-insensitive mode to optimize the lookup calls towards the data provider.

The *lower-case* statement can be used in: *leaf* and *leaf-list*.

tailf:meta-data *value*

Extra meta information attached to the node. The instance data part of this information is accessible using MAAPI. It is also printed in communication with CLI NEDs, but is not visible to normal users of the CLI.

```
To CLI NEDs, the output will be printed as comments like this:  
! meta-data :: /ncs:devices/device{xyz}/config/xyz:AA :: A_STRING
```

The schema information is available to the ConfD C-API through the `confd_cs_node` struct, and to the JSON-RPC API through `get-schema`.

Note: Can't be used on key leafs.

The *meta-data* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *tailf:symlink*, and *refine*.

The following substatements can be used:

tailf:meta-value This statement contains a string value for the meta data key.

The output from the CLI to CLI NEDs will be similar to comments like this: `! meta-data :: /ncs:devices/device{xyz}/config/xyz:AA :: A_KEY :: A_VALUE`

tailf:ncs-device-type *type*

Internal extension to tell NCS what type of device the data model is used for.

The *ncs-device-type* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, *refine*, and *module*.

tailf:ned-data *path-expression*

Dynamic meta information to be added by the NCS device manager.

In the cases where NCS can't provide the complete 'to' and 'from' transactions to the NED to read from (most notably when using the commit queue) this annotation can be used to tell the NCS device manager to save part of the 'to' and / or 'from' transaction so that the NED will be able to read from these parts as needed.

The 'path-expression' will be used as an XPath filter to indicate which data will be preserved. Use the 'transaction' substatement to choose which transaction to apply the filter on. The context node of the XPath filter is always the instance data node corresponding to the schema node where the 'ned-data' extension is added.

Note that the filter will only be applied if the node that has this annotation is in the diffset of the transaction. The 'operation' substatement can be used to further limit when the filter should be applied.

The *ned-data* statement can be used in: *container*, *list*, *leaf*, *leaf-list*, and *refine*.

The following substatements can be used:

tailf:transaction

tailf:xpath-root

tailf:operation

tailf:ned-default-handling *mode*

This statement can only be used in NEDs for devices that have irregular handling of defaults. It sets a special default handling mode for the leaf, regardless of the device's native default handling mode.

The *ned-default-handling* statement can be used in: *leaf*.

tailf:ned-ignore-compare-config

Typically used for ignoring device encrypted leafs in the compare-config output.

The *ned-ignore-compare-config* statement can be used in: *leaf*.

tailf:no-dependency

This optional statements can be used to explicitly say that a 'must' expression or a validation function is evaluated at every commit. Use this with care, since the overall performance of the system is impacted if this statement is used.

The *no-dependency* statement can be used in: *must* and *tailf:validate*.

tailf:no-leafref-check

This statement can be used to let 'leafref' type statements reference non-existing leafs. While similar to the 'tailf:non-strict-leafref' statement, this does not allow reference from config to non-config.

The *no-leafref-check* statement can be used in: *type*.

tailf:non-strict-leafref

This statement can be used in leafs and leaf-lists similar to 'leafref', but allows reference to non-existing leafs, and allows reference from config to non-config.

This statement takes no argument, but expects the core YANG statement 'path' as a substatement. The function 'deref' cannot be used in the path, since it works on nodes of type leafref only.

The type of the leaf or leaf-list must be exactly the same as the type of the target.

This statement can be viewed as a substitute for a standard 'require-instance false' on leafrefs, which isn't allowed.

The CLI uses this statement to provide completion with existing values, and the WebUI uses it to provide a drop-down box with existing values.

The *non-strict-leafref* statement can be used in: *leaf* and *leaf-list*.

tailf:operation op

Only evaluate the XPath filter when the operation matches.

tailf:override-auto-dependencies

This optional statement can be used to instruct the compiler to use the provided tailf:dependency statements instead of the dependencies that the compiler calculates from the expression.

Use with care, and only if you are sure that the provided dependencies are correct.

The *override-auto-dependencies* statement can be used in: *must* and *when*.

tailf:path-filters *value*

Used for type 'instance-identifier' only.

The argument is a space separated list of absolute or relative XPath expressions.

This statement declares that the instance-identifier value must match one of the specified paths, according to the following rules:

1. each XPath expression is evaluated, and returns a node set.
2. if there is no 'tailf:no-subtree-match' statement, the instance-identifier matches if it refers to a node in this node set, or if it refers to any descendant node of this node set.

3. if there is a 'tailf:no-subtree-match' statement, the instance-identifier matches if it refers to a node in this node set.

For example:

The value /a/b[key='k1']/c matches the XPath expression /a/b[key='k1']/c.

The value /a/b[key='k1']/c matches the XPath expression /a/b/c.

The value /a/b[key='k1']/c matches the XPath expression /a/b, if there is no 'tailf:no-subtree-match' statement.

The value /a/b[key='k1'] matches the XPath expression /a/b, if there is a 'tailf:no-subtree-match' statement.

The *path-filters* statement can be used in: *type*.

The following substatements can be used:

tailf:no-subtree-match See tailf:path-filters.

tailf:secondary-index *name*

This statement creates a secondary index with a given name in the parent list. The secondary index can be used to control the displayed sort order of the instances of the list.

Read more about sort order in 'The ConfD Command-Line Interface (CLI)' chapters in the User Guide, confd_lib_dp(3), and confd_lib_maapi(3).

NOTE: Currently secondary-index is not supported for config false data stored in CDB.

The *secondary-index* statement can be used in: *list*.

The following substatements can be used:

tailf:index-leafs This statement contains a space separated list of leaf names. Each such leaf must be a direct child to the list. The secondary index is kept sorted according to the values of these leafs.

tailf:sort-order

tailf:display-default-order Specifies that the list should be displayed sorted according to this secondary index in the show command.

If the list has more than one secondary index, 'display-default-order' must be present in one index only.

Used in J-, I- and C-style CLIs and WebUI.

tailf:snmp-delete-value *value*

This statement is used to define a value to be used in SNMP to delete an optional leaf. The argument to this statement is the special value. This special value must not be part of the value space for the YANG leaf.

If the optional leaf does not exist, reading it over SNMP returns 'noSuchInstance', unless the statement 'tailf:snmp-send-delete-value' is used, in which case the same value as used to delete the node is returned.

For example, the YANG leaf:

```
leaf opt-int {
  type int32 {
    range '1..255';
  }
}
```

```

tailf:snmp-delete-value 0 {
    tailf:snmp-send-delete-value;
}

```

can be mapped to a SMI object with syntax:

SYNTAX Integer32 (0..255)

Setting such an object to '0' over SNMP will delete the node from the datastore. If the node does not exist, reading it over SNMP will return '0'.

The *snmp-delete-value* statement can be used in: *leaf*.

The following substatements can be used:

tailf:snmp-send-delete-value See *tailf:snmp-delete-value*.

tailf:snmp-exclude-object

Used when an SNMP MIB is generated from a YANG module, using the `--generate-oids` option to `confdc`.

If this statement is present, `confdc` will exclude this object from the resulting MIB.

The *snmp-exclude-object* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, and *refine*.

tailf:snmp-lax-type-check *value*

Normally, the ConfD MIB compiler checks that the data type of an SNMP object matches the data type of the corresponding YANG leaf. If both objects are writable, the data types need to precisely match, but if the SNMP object is read-only, or if `snmp-lax-type-check` is set to 'true', the compiler accepts the object if the SNMP type's value space is a superset of the YANG type's value space.

If `snmp-lax-type-check` is true and the MIB object is writable, the SNMP agent will reject values outside the YANG data type range in runtime.

The *snmp-lax-type-check* statement can be used in: *leaf*.

tailf:snmp-mib-module-name *name*

Used when the YANG module is mapped to an SNMP module.

Specifies the name of the SNMP MIB module where the SNMP objects are defined.

This property is inherited by all child nodes.

The *snmp-mib-module-name* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *module*, and *refine*.

tailf:snmp-name *name*

Used when the YANG module is mapped to an SNMP module.

When the parent node is mapped to an SNMP object, this statement specifies the name of the SNMP object.

If the parent node is mapped to multiple SNMP objects, this statement can be given multiple times. The first statement specifies the primary table.

In a list, the argument is interpreted as:

[MIB-MODULE-NAME:]TABLE-NAME

For a leaf representing a table column, it is interpreted as:

[[MIB-MODULE-NAME:]TABLE-NAME:]NAME

For a leaf representing a scalar variable, it is interpreted as:

[MIB-MODULE-NAME:]NAME

If a YANG list is mapped to multiple SNMP tables, each such SNMP table must be specified with a 'tailf:snmp-name' statement. If the table is defined in another MIB than the MIB specified in 'tailf:snmp-mib-module-name', the MIB name must be specified in this argument.

A leaf in a list that is mapped to multiple SNMP tables must specify the name of the table it is mapped to if it is different from the primary table.

In the following example, a single YANG list 'interface' is mapped to the MIB tables ifTable, ifXTable, and ipv4InterfaceTable:

```
list interface {
  key index;
  tailf:snmp-name 'ifTable'; // primary table
  tailf:snmp-name 'ifXTable';
  tailf:snmp-name 'IP-MIB:ipv4InterfaceTable';

  leaf index {
    type int32;
  }
  leaf description {
    type string;
    tailf:snmp-name 'ifDescr'; // mapped to primary table
  }
  leaf name {
    type string;
    tailf:snmp-name 'ifXTable:ifName';
  }
  leaf ipv4-enable {
    type boolean;
    tailf:snmp-name
      'IP-MIB:ipv4InterfaceTable:ipv4InterfaceEnableStatus';
  }
  ...
}
```

When emitting a mib from yang, enum labels are used as-is if they follow the SMI rules for labels (no '.' or '_' characters and beginning with a lowercase letter). Any label that doesn't satisfy the SMI rules will be converted as follows:

An initial uppercase character will be downcased.

If the initial character is not a letter it will be prepended with an 'a'.

Any '.' or '_' characters elsewhere in the label will be substituted with '-' characters.

In the resulting label, any multiple '-' character sequence will be replaced with a single '-' character.

If this automatic conversion is not suitable, snmp-name can be used to specify the label to use when emitting a MIB.

The *snmp-name* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, *enum*, and *refine*.

tailf:snmp-ned-accessible-column *leaf-name*

The name or subid number of an accessible column that is instantiated in all table entries in a table. The column does not have to be writable. The SNMP NED will use this column when it uses GET-NEXT to loop through the list entries, and when doing existence tests.

If this column is not given, the SNMP NED uses the following algorithm:

1. If there is a RowStatus column, it will be used. 2. If an INDEX leaf is accessible, it will be used. 3. Otherwise, use the first accessible column returned by the SNMP agent.

The *snmp-ned-accessible-column* statement can be used in: *list*.

tailf:snmp-ned-delete-before-create

This statement is used in a list to make the SNMP NED always send deletes before creates. Normally, creates are sent before deletes.

The *snmp-ned-delete-before-create* statement can be used in: *list*.

tailf:snmp-ned-modification-dependent

This statement is used on all columns in a table that require the usage of the column marked with tailf:snmp-ned-set-before-row-modification.

This statement can be used on any column in a table where one leaf is marked with tailf:snmp-ned-set-before-row-modification, or a table that AUGMENTS such a table, or a table with a foreign index in such a table.

The *snmp-ned-modification-dependent* statement can be used in: *leaf*.

tailf:snmp-ned-recreate-when-modified

This statement is used in a list to make the SNMP NED delete and recreate the row when a column in the row is modified.

The *snmp-ned-recreate-when-modified* statement can be used in: *list*.

tailf:snmp-ned-set-before-row-modification *value*

If this statement is present on a leaf, it tells the SNMP NED that if a column in the row is modified, and it is marked with 'tailf:snmp-ned-modification-dependent', then the column marked with 'tailf:snmp-ned-set-before-modification' needs to be set to <value> before the other column is modified. After all such columns have been modified, the column marked with 'tailf:snmp-ned-set-before-modification' is reset to its initial value.

The *snmp-ned-set-before-row-modification* statement can be used in: *leaf*.

tailf:snmp-oid *oid*

Used when the YANG module is mapped to an SNMP module.

If this statement is present as a direct child to 'module', it indicates the top level OID for the module.

When the parent node is mapped to an SNMP object, this statement specifies the OID of the SNMP object. It may be either a full OID or just a suffix (a period, followed by an integer). In the latter case, a full OID must be given for some ancestor element.

NOTE: when this statement is set in a list, it refers to the OID of the corresponding table, not the table entry.

The *snmp-oid* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *tailf:symlink*, *module*, and *refine*.

tailf:snmp-row-status-column *value*

Used when an SNMP module is generated from the YANG module.

When the parent list node is mapped to an SNMP table, this statement specifies the column number of the generated RowStatus column. If it is not specified, the generated RowStatus column will be the last in the table.

The *snmp-row-status-column* statement can be used in: *list* and *refine*.

tailf:sort-order *how*

This statement can be used for 'ordered-by system' lists and leaf-lists only. It indicates in which way the list entries are sorted.

The *sort-order* statement can be used in: *list*, *leaf-list*, and *tailf:secondary-index*.

tailf:sort-priority *value*

This extension takes an integer parameter specifying the order and can be placed on leaves, containers, lists and leaf-lists. When showing, or getting configuration, leaf values will be returned in order of increasing sort-priority.

The default sort-priority is 0.

The *sort-priority* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, and *refine*.

tailf:step *value*

Used to further restrict the range of integer and decimal types. The argument is a positive integer or decimal value greater than zero. The allowed values for the type is further restricted to only those values that matches the expression:

'low' + n * 'step'

where 'low' is the lowest allowed value in the range, n is a non-negative integer.

For example, the following type:

```
type int32 {  
    range '-2 .. 9' {  
        tailf:step 3;  
    }  
}
```

has the value space { -2, 1, 4, 7 }

The *step* statement can be used in: *range*.

tailf:structure *name*

Internal extension to define a data structure without any semantics attached.

The *structure* statement can be used in: *module* and *submodule*.

tailf:suppress-echo *value*

If this statement is set to 'true', leafs of this type will not have their values echoed when input in the webui or when the CLI prompts for the value. The value will also not be included in the audit log in clear text but will appear as ***.

The *suppress-echo* statement can be used in: *typedef*, *leaf*, and *leaf-list*.

tailf:symlink *name*

DEPRECATED: Use *tailf:link* instead. There are no plans to remove *tailf:symlink*.

This statement defines a 'symbolic link' from a node to some other node. The argument is the name of the new node, and the mandatory substatement 'tailf:path' points to the node which is linked to.

The *symlink* statement can be used in: *list*, *container*, *module*, *submodule*, *augment*, and *case*.

The following substatements can be used:

tailf:alt-name

tailf:cli-add-mode

tailf:cli-allow-join-with-key

tailf:cli-allow-join-with-value

tailf:cli-allow-key-abbreviation

tailf:cli-allow-range

tailf:cli-allow-wildcard

tailf:cli-autowizard

tailf:cli-boolean-no

tailf:cli-break-sequence-commands

tailf:cli-column-align

tailf:cli-column-stats

tailf:cli-column-width

tailf:cli-compact-stats

tailf:cli-compact-syntax

tailf:cli-completion-actionpoint

tailf:cli-custom-error

tailf:cli-custom-range

tailf:cli-custom-range-actionpoint

tailf:cli-custom-range-enumerator

tailf:cli-delayed-auto-commit

tailf:cli-delete-container-on-delete

tailf:cli-delete-when-empty

tailf:cli-diff-dependency

tailf:cli-disabled-info

tailf:cli-disallow-value
tailf:cli-display-empty-config
tailf:cli-display-separated
tailf:cli-drop-node-name
tailf:cli-no-keyword
tailf:cli-enforce-table
tailf:cli-embed-no-on-delete
tailf:cli-exit-command
tailf:cli-explicit-exit
tailf:cli-expose-key-name
tailf:cli-expose-ns-prefix
tailf:cli-flat-list-syntax
tailf:cli-flatten-container
tailf:cli-full-command
tailf:cli-full-no
tailf:cli-full-show-path
tailf:cli-hide-in-submode
tailf:cli-ignore-modified
tailf:cli-incomplete-command
tailf:cli-incomplete-no
tailf:cli-incomplete-show-path
tailf:cli-instance-info-leafs
tailf:cli-key-format
tailf:cli-list-syntax
tailf:cli-min-column-width
tailf:cli-mode-name
tailf:cli-mode-name-actionpoint
tailf:cli-multi-value
tailf:cli-multi-word-key
tailf:cli-multi-line-prompt
tailf:cli-no-key-completion
tailf:cli-no-match-completion

tailf:cli-no-name-on-delete
tailf:cli-no-value-on-delete
tailf:cli-oper-info
tailf:cli-optional-in-sequence
tailf:cli-prefix-key
tailf:cli-preformatted
tailf:cli-range-delimiters
tailf:cli-range-list-syntax
tailf:cli-recursive-delete
tailf:cli-remove-before-change
tailf:cli-reset-container
tailf:cli-run-template
tailf:cli-run-template-enter
tailf:cli-run-template-footer
tailf:cli-run-template-legend
tailf:cli-sequence-commands
tailf:cli-show-config
tailf:cli-show-no
tailf:cli-show-order-tag
tailf:cli-show-order-taglist
tailf:cli-show-template
tailf:cli-show-template-enter
tailf:cli-show-template-footer
tailf:cli-show-template-legend
tailf:cli-show-with-default
tailf:cli-strict-leafref
tailf:cli-suppress-key-abbreviation
tailf:cli-suppress-key-sort
tailf:cli-suppress-list-no
tailf:cli-suppress-mode
tailf:cli-suppress-no
tailf:cli-suppress-range

tailf:cli-suppress-shortenabled
tailf:cli-suppress-show-conf-path
tailf:cli-suppress-show-match
tailf:cli-suppress-show-path
tailf:cli-suppress-silent-no
tailf:cli-suppress-validation-warning-prompt
tailf:cli-suppress-wildcard
tailf:cli-table-footer
tailf:cli-table-legend
tailf:cli-trim-default
tailf:cli-value-display-template
tailf:display-when
tailf:hidden

tailf:inherit-set-hook This statement specifies that a 'tailf:set-hook' statement should survive through symlinks. If set to true a set hook gets called as soon as the value is set via a symlink but also during commit. The normal behaviour is to only call the set hook during commit time.

tailf:info

tailf:info-html

tailf:path This statement specifies which node a symlink points to.

The textual format of a symlink is an XPath absolute location path. If the target lies within lists, all keys must be specified. A key either has a value, or is a reference to a key in the path of the source node, using the function `current()` as starting point for an XPath location path. For example:

`/a/b[k1='paul'][k2=current()../k]/c`

tailf:snmp-exclude-object

tailf:snmp-name

tailf:snmp-oid

tailf:sort-priority

tailf:transaction *direction*

Which transaction that the result of the XPath filter will be applied to, when set to 'both' it will apply to both the 'to' and the 'from' transaction.

tailf:typepoint *id*

If a typedef, leaf, or leaf-list has a 'typepoint' statement, a user-defined type is specified, as opposed to a derivation or specification of an existing type. The implementation of a user-defined type must be provided in the form of a shared object with C callback functions that is loaded into the ConfD daemon at startup time. Read more about user-defined types in the `confd_types(3)` manual page.

The argument defines the ID associated with a typepoint. This ID is provided by the shared object, and used by the ConfD daemon to locate the implementation of a specific user-defined type.

The *typepoint* statement can be used in: *typedef*, *leaf*, and *leaf-list*.

tailf:unique-selector *context-path*

The standard YANG statement 'unique' can be used to check for uniqueness within a single list only. Specifically, it cannot be used to check for uniqueness of leafs within a sublist.

For example:

```
container a {
  list b {
    ...
    unique 'server/ip server/port';
    list server {
      ...
      leaf ip { ... };
      leaf port { ... };
    }
  }
}
```

The unique expression above is not legal. The intention is that there must not be any two 'server' entries in any 'b' with the same combination of ip and port. This would be illegal:

```
<a> <b> <name>b1</name> <server> <ip>10.0.0.1</ip> <port>80</port> </server> </b> <b>
<name>b2</name> <server> <ip>10.0.0.1</ip> <port>80</port> </server> </b> </a>
```

With 'tailf:unique-selector' and 'tailf:unique-leaf', this kind of constraint can be defined.

The argument to 'tailf:unique-selector' is an XPath descendant location path (matches the rule 'descendant-schema-nodeid' in RFC 6020). The first node in the path **MUST** be a list node, and it **MUST** be defined in the same module as the tailf:unique-selector. For example, the following is illegal:

```
module y {
  ...
  import x {
    prefix x;
  }
  tailf:unique-selector '/x:server' { // illegal
    ...
  }
}
```

For each instance of the node where the selector is defined, it is evaluated, and for each node selected by the selector, a tuple is constructed by evaluating the 'tailf:unique-leaf' expression. All such tuples must be unique. If a 'tailf:unique-leaf' expression refers to a non-existing leaf, the corresponding tuple is ignored.

In the example above, the unique expression can be replaced by:

```
container a {
  tailf:unique-selector 'b/server' {
    tailf:unique-leaf 'ip';
    tailf:unique-leaf 'port';
  }
  list b {
    ...
  }
}
```

For each container 'a', the XPath expression 'b/server' is evaluated. For each such server, a 2-tuple is constructed with the 'ip' and 'port' leafs. Each such 2-tuple is guaranteed to be unique.

The *unique-selector* statement can be used in: *module*, *submodule*, *grouping*, *augment*, *container*, and *list*.

The following substatements can be used:

tailf:unique-leaf See 'tailf:unique-selector' for a description of how this statement is used.

The argument is an XPath descendant location path (matches the rule 'descendant-schema-nodeid' in RFC 6020), and it MUST refer to a leaf.

tailf:validate *id*

Identifies a validation callback which is invoked when a configuration value is to be validated. The callback validates a value and typically checks it towards other values in the data store. Validation callbacks are used when the YANG built-in validation constructs ('must', 'unique') are not expressive enough.

Callbacks use the API described in `confd_lib_maapi(3)` to access whatever other configuration values needed to perform the validation.

Validation callbacks are typically assigned to individual nodes in the data model, but it may be feasible to use a single validation callback on a root node. In that case the callback is responsible for validation of all values and their relationships throughout the data store.

The 'validate' statment should in almost all cases have a 'tailf:dependency' substatement. If such a statement is not given, the validate function is evaluated at every commit, leading to overall performance degradation.

If the 'validate' statement is defined in a 'must' statement, validation callback is called instead of evaluating the must expression. This is useful if the evaluation of the must statement uses too much resources, and the condition expressed with the must statement is easier to check with a validation callback function.

The *validate* statement can be used in: *leaf*, *leaf-list*, *list*, *container*, *grouping*, *refine*, and *must*.

The following substatements can be used:

tailf:call-once This optional statement can be used only if the parent statement is a list. If 'call-once' is 'true', the validation callback is only called once even though there exists many list entries in the data store. This is useful if we have a huge amount of instances or if values assigned to each instance have to be validated in comparison with its siblings.

tailf:dependency

tailf:opaque Defines an opaque string which is passed to the callback function in the context.

tailf:internal For internal ConfD / NCS use only.

tailf:priority This extension takes an integer parameter specifying the order validation code will be evaluated, in order of increasing priority.

The default priority is 0.

tailf:value-length *value*

Used only for the types: `yang:object-identifier` `yang:object-identifier-128` `yang:phys-address` `yang:hex-string` `tailf:hex-list` `tailf:octet-list` `xs:hexBinary`

This type restriction is used to limit the length of the value-space value of the type. Note that since all these types are derived from 'string', the standard 'length' statement restricts the lexical representation of the value.

The argument is a length expression string, with the same syntax as for the standard YANG 'length' statement.

The *value-length* statement can be used in: *type*.

tailf:writable *value*

This extension makes operational data (i.e., config false data) writable. Only valid for leafs.

The *writable* statement can be used in: *leaf*.

tailf:xpath-root *value*

Internal extension to 'chroot' XPath expressions

The *xpath-root* statement can be used in: *must*, *when*, *path*, *tailf:display-when*, *tailf:cli-diff-dependency*, *tailf:cli-diff-before*, *tailf:cli-diff-delete-before*, *tailf:cli-diff-set-before*, *tailf:cli-diff-create-before*, *tailf:cli-diff-modify-before*, *tailf:cli-diff-after*, *tailf:cli-diff-delete-after*, *tailf:cli-diff-set-after*, *tailf:cli-diff-create-after*, and *tailf:cli-diff-modify-after*.

YANG TYPES

aes-cfb-128-encrypted-string

The aes-cfb-128-encrypted-string works exactly like des3-cbc-encrypted-string but AES/128bits in CFB mode is used to encrypt the string. The prefix for encrypted values is '\$8\$'.

des3-cbc-encrypted-string

The des3-cbc-encrypted-string type automatically encrypts a value adhering to this type using DES in CBC mode followed by a base64 conversion. If the value isn't encrypted already, that is.

This is best explained using an example. Suppose we have a leaf:

```
leaf enc {  
    type tailf:des3-cbc-encrypted-string;  
}
```

A valid configuration is:

```
<enc>$0$In god we trust.</enc>
```

The '\$0\$' prefix signals that this is plain text. When a plain text value is received by the server, the value is DES3/Base64 encrypted, and the string '\$7\$' is prepended. The resulting string is stored in the configuration data store.

When a value of this type is read, the encrypted value is always returned. In the example above, the following value could be returned:

```
<enc>$7$Qxxsn8BVzxphCdfqRwZm6noKKmt0QoSWnRnhcXqocg=</enc>
```

If a value starting with '\$7\$' is received, the server knows that the value is already encrypted, and stores it as is in the data store.

A value adhering to this type must have a '\$0\$' or a '\$7\$' prefix.

ConfD uses a configurable set of encryption keys to encrypt the string. For details, see 'encryptedStrings' in the confd.conf(5) manual page.

hex-list

DEPRECATED: Use yang:hex-string instead. There are no plans to remove tailf:hex-list.

A list of colon-separated hexa-decimal octets e.g. '4F:4C:41:71'.

The statement tailf:value-length can be used to restrict the number of octets. Note that using the 'length' restriction limits the number of characters in the lexical representation.

ip-address-and-prefix-length

The ip-address-and-prefix-length type represents a combination of an IP address and a prefix length and is IP version neutral. The format of the textual representations implies the IP version.

ipv4-address-and-prefix-length

The ipv4-address-and-prefix-length type represents a combination of an IPv4 address and a prefix length. The prefix length is given by the number following the slash character and must be less than or equal to 32.

ipv6-address-and-prefix-length

The ipv6-address-and-prefix-length type represents a combination of an IPv6 address and a prefix length. The prefix length is given by the number following the slash character and must be less than or equal to 128.

md5-digest-string

The md5-digest-string type automatically computes a MD5 digest for a value adhering to this type.

This is best explained using an example. Suppose we have a leaf:

```
leaf key {  
    type tailf:md5-digest-string;  
}
```

A valid configuration is:

```
<key>$0$In god we trust.</key>
```

The '\$0\$' prefix signals that this is plain text. When a plain text value is received by the server, an MD5 digest is calculated, and the string '\$1\$<salt>\$' is prepended to the result, where <salt> is a random eight character salt used to generate the digest. This value is stored in the configuration data store.

When a value of this type is read, the computed MD5 value is always returned. In the example above, the following value could be returned:

```
<key>$1$fB$ndk2z/PIS0S1SvzWLqTJb.</key>
```

If a value starting with '\$1\$' is received, the server knows that the value already represents an MD5 digest, and stores it as is in the data store.

A value adhering to this type must have a '\$0\$' or a '\$1\$<salt>\$' prefix.

If a default value is specified, it must have a '\$1\$<salt>\$' prefix.

The digest algorithm used is the same as the md5 crypt function used for encrypting passwords for various UNIX systems, see e.g. <http://www.freebsd.org/cgi/cvsweb.cgi/~checkout/~src/lib/libcrypt/crypt.c>

octet-list

A list of dot-separated octets e.g. '192.168.255.1.0'.

The statement `tailf:value-length` can be used to restrict the number of octets. Note that using the 'length' restriction limits the number of characters in the lexical representation.

sha-256-digest-string

The sha-256-digest-string type automatically computes a SHA-256 digest for a value adhering to this type.

A value of this type matches one of the forms:

`0<clear text password> 5<salt>$<password hash> 5rounds=<number>$<salt>$<password hash>`

The '\$0\$' prefix signals that this is plain text. When a plain text value is received by the server, a SHA-256 digest is calculated, and the string '\$5\$<salt>\$' is prepended to the result, where <salt> is a random 16 character salt used to generate the digest. This value is stored in the configuration data store. The algorithm can be tuned via the `/confdConfig/cryptHash/rounds` parameter, which if set to a number other than the default will cause '\$5\$rounds=<number>\$<salt>\$' to be prepended instead of only '\$5\$<salt>\$'.

If a value starting with '\$5\$' is received, the server knows that the value already represents a SHA-256 digest, and stores it as is in the data store.

If a default value is specified, it must have a '\$5\$' prefix.

The digest algorithm used is the same as the SHA-256 crypt function used for encrypting passwords for various UNIX systems, see e.g. <http://www.akkadia.org/drepper/SHA-crypt.txt>

sha-512-digest-string

The sha-512-digest-string type automatically computes a SHA-512 digest for a value adhering to this type.

A value of this type matches one of the forms:

`0<clear text password> 6<salt>$<password hash> 6rounds=<number>$<salt>$<password hash>`

The '\$0\$' prefix signals that this is plain text. When a plain text value is received by the server, a SHA-512 digest is calculated, and the string '\$6\$<salt>\$' is prepended to the result, where <salt> is a random 16 character salt used to generate the digest. This value is stored in the configuration data store. The algorithm can be tuned via the `/confdConfig/cryptHash/rounds` parameter, which if set to a number other than the default will cause '\$6\$rounds=<number>\$<salt>\$' to be prepended instead of only '\$6\$<salt>\$'.

If a value starting with '\$6\$' is received, the server knows that the value already represents a SHA-512 digest, and stores it as is in the data store.

If a default value is specified, it must have a '\$6\$' prefix.

The digest algorithm used is the same as the SHA-512 crypt function used for encrypting passwords for various UNIX systems, see e.g. <http://www.akkadia.org/drepper/SHA-crypt.txt>

size

A value that represents a number of bytes. An example could be `S1G8M7K956B`; meaning 1GB + 8MB + 7KB + 956B = 1082138556 bytes. The value must start with an S. Any byte magnifier can be left out, e.g. `S1K1B` equals 1025 bytes. The order is significant though, i.e. `S1B56G` is not a valid byte size.

In ConfD, a 'size' value is represented as an uint64.

XPATH FUNCTIONS

This section describes XPath functions that can be used for example in "must" expressions in YANG modules.

node-set **deref**(*node-set*)

The `deref()` function follows the reference defined by the first node in document order in the argument node-set, and returns the nodes it refers to.

If the first argument node is an instance-identifier, the function returns a node-set that contains the single node that the instance identifier refers to, if it exists. If no such node exists, an empty node-set is returned.

If the first argument node is a leafref, the function returns a node-set that contains the nodes that the leafref refers to.

If the first argument node is of any other type, an empty node-set is returned.

bool **re-match**(*string*, *string*)

The `re-match()` function returns `true` if the string in the first argument matches the regular expression in the second argument; otherwise it returns `false`.

For example: `re-match('1.22.333', '\d{1,3}\.\d{1,3}\.\d{1,3}')` returns `true`. To count all logical interfaces called `eth0.number`: `count(/sys/ifc[re-match(name, 'eth0\.\d+')])`.

The regular expressions used are the XML Schema regular expressions, as specified by W3C in <http://www.w3.org/TR/xmlschema-2/#regexs>. Note that this includes implicit anchoring of the regular expression at the head and tail, i.e. if you want to match an interface that has a name that starts with 'eth' then the regular expression must be `'eth.*'`.

number **string-compare**(*string*, *string*)

The `string-compare()` function returns -1, 0, or 1 depending on whether the value of the string of the first argument is respectively less than, equal to, or greater than the value of the string of the second argument.

number **compare**(*Expression*, *Expression*)

The `compare()` function returns -1, 0, or 1 depending on whether the value of the first argument is respectively less than, equal to, or greater than the value of the second argument.

The expressions are evaluated in a special way: If they both are XPath constants they are compared using the `string-compare()` function. But, more interestingly, if the expressions results in node-sets with at least one node, and that node is an existing leaf that leafs value is compared with the other expression, and if the other expression is a constant that expression is converted to an internal value with the same type as the expression that resulted in a leaf. Thus making it possible to order values based on the internal representation rather than the string representation. For example, given a leaf:

```
leaf foo {
```

```

type enumeration {
    enum ccc;
    enum bbb;
    enum aaa;
}

```

it would be possible to call `compare(foo, 'bbb')` (which, for example, would return -1 if `foo='ccc'`). Or to have a must expression like this: `must "compare(., 'bbb') >= 0"`; which would require `foo` to be set to 'bbb' or 'aaa'.

If one of the expressions result in an empty node-set, a non-leaf node, or if the constant can't be converted to the other expressions type then NaN is returned.

number **min**(*node-set*)

Returns the numerically smallest number in the node-set, or NaN if the node-set is empty.

number **max**(*node-set*)

Returns the numerically largest number in the node-set, or NaN if the node-set is empty.

number **avg**(*node-set*)

Returns the numerical average of the node-set, or NaN if the node-set is empty, or if any numerical conversion of a node failed.

number **band**(*number*, *number*)

Returns the result of bitwise AND:ing the two numbers. Unless the numbers are integers NaN will be returned.

number **bor**(*number*, *number*)

Returns the result of bitwise OR:ing the two numbers. Unless the numbers are integers NaN will be returned.

number **bxor**(*number*, *number*)

Returns the result of bitwise Exclusive OR:ing the two numbers. Unless the numbers are integers NaN will be returned.

number **bnot**(*number*)

Returns the result of bitwise NOT on number. Unless the number is an integer NaN will be returned.

node-set **sort-by**(*node-set*, *string*)

The `sort-by()` function makes it possible to order a node-set according to a secondary index (see the [tailf:secondary-index](#) extension). The first argument must be an expression that evaluates to a node-set, where the nodes in the node-set are all list instances of the same list. The second argument must be the name of an existing secondary index on that list. For example given the YANG model:

```

container sys {
    list host {
        key name;
        unique number;
        tailf:secondary-index number {
            tailf:index-leafs "number";
        }
        leaf name {
            type string;
        }
        leaf number {
            type uint32;
            mandatory true;
        }
        leaf enabled {
            type boolean;
            default true;
        }
    }
    ...
}

```

```
}
```

The expression `sort-by(/sys/host , "number")` would result in all hosts, sorted by their number. And the expression, `sort-by(/sys/host[enabled= 'true'] , "number")` would result in all enabled hosts, sorted by number. Note also that since the function returns a node-set it is also legal to add location steps to the result. I.e. the expression `sort-by(/sys/host[enabled= 'true'] , "number")/name` results in all host names sorted by the hosts number.

SEE ALSO

`tailf_yang_cli_extensions(5)`
The NSO User Guide
`confdc(1)`

Tail-f YANG CLI extensions

Confdc compiler