

---

# **NCClient Documentation**

*Release 0.1.1a*

**Shikhar Bhushan**

May 17, 2009



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>User documentation</b>	<b>3</b>
2.1	manager module . . . . .	3
2.2	capabilities module . . . . .	4
2.3	content module . . . . .	5
2.4	transport module . . . . .	7
2.5	operations module . . . . .	9
<b>3</b>	<b>Extending NCClient</b>	<b>15</b>
	<b>Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



# INTRODUCTION

NCClient is a Python library for NETCONF clients. NETCONF is a network management protocol defined in [RFC 4741](#). It is meant for Python 2.6+ (not Python 3 yet, though).

The features of NCClient include:

- Request pipelining.
- (A)synchronous RPC requests.
- Keeps XML out of the way unless really needed.
- Supports all operations and capabilities defined in [RFC 4741](#).
- Extensible. New transport mappings and capabilities/operations can be easily added.

The best way to introduce is of course, through a simple code example:

```
from ncclient import manager

with manager.connect_ssh('host', 'username') as m:
    assert(":url" in manager.server_capabilities)
    with m.locked('running'):
        m.copy_config(source="running", target="file://new_checkpoint.conf")
        m.copy_config(source="file://old_checkpoint.conf", target="running")
```

It is recommended to use the high-level Manager API where possible. It exposes almost all of the functionality.



# USER DOCUMENTATION

## 2.1 manager module

### 2.1.1 Dealing with RPC errors

These constants define what `Manager` does when an `<rpc-error>` element is encountered in a reply.

**RAISE\_ALL**

Raise all `RPCError`

**RAISE\_ERR**

Only raise when `error-severity` is “error” i.e. no warnings

**RAISE\_NONE**

Don’t raise any

### 2.1.2 Manager instances

`Manager` instances are created by the `connect()` family of factory functions. Currently only `connect_ssh()` is available.

**connect** (*\*args, \*\*kws*)

Same as `connect_ssh()`

**connect\_ssh** (*\*args, \*\*kws*)

Connect to NETCONF server over SSH. See `SSHSession.connect()` for function signature.

**class Manager** (*session*)

API for NETCONF operations. Currently only supports making synchronous RPC requests.

It is also a context manager, so a `Manager` instance can be used with the `with` statement. The session is closed when the context ends.

**set\_rpc\_error\_action** (*action*)

Specify the action to take when an `<rpc-error>` element is encountered.

**Parameter** *action* – one of `RAISE_ALL`, `RAISE_ERR`, `RAISE_NONE`

**get** (*\*args, \*\*kws*)

See `Get.request()`

**get\_config** (*\*args, \*\*kws*)

See `GetConfig.request()`

**edit\_config** (*\*args, \*\*kws*)

See `EditConfig.request()`

**copy\_config** (*\*args, \*\*kws*)

See `CopyConfig.request()`

**validate** (*\*args, \*\*kws*)

**See** `GetConfig.request()`

**commit** (*\*args*, *\*\*kwds*)  
**See** `Commit.request()`

**discard\_changes** (*\*args*, *\*\*kwds*)  
**See** `DiscardChanges.request()`

**delete\_config** (*\*args*, *\*\*kwds*)  
**See** `DeleteConfig.request()`

**lock** (*\*args*, *\*\*kwds*)  
**See** `Lock.request()`

**unlock** (*\*args*, *\*\*kwds*)  
**See** `DiscardChanges.request()`

**close\_session** (*\*args*, *\*\*kwds*)  
**See** `CloseSession.request()`

**kill\_session** (*\*args*, *\*\*kwds*)  
**See** `KillSession.request()`

**locked** (*target*)  
Returns a context manager for the *with* statement.  
**Parameter** *target* (*string*) – name of the datastore to lock  
**Return type** `operations.LockContext`

**close** ()  
Closes the NETCONF session. First does `<close-session>` RPC.

**client\_capabilities**  
`Capabilities` object for client

**server\_capabilities**  
`Capabilities` object for server

**session\_id**  
`<session-id>` as assigned by NETCONF server

**connected**  
Whether currently connected to NETCONF server

## 2.2 capabilities module

### CAPABILITIES

`Capabilities` object representing the capabilities currently supported by NCClient

#### class `Capabilities` (*capabilities*)

Represents the set of capabilities for a NETCONF client or server. Initialised with a list of capability URI's.

Presence of a capability can be checked with the *in* operations. In addition to the URI, for capabilities of the form `urn:ietf:params:netconf:capability:$name:$version` their shorthand can be used as a key. For example, for `urn:ietf:params:netconf:capability:candidate:1.0` the shorthand would be `:candidate`. If version is significant, use `:candidate:1.0` as key.

#### **add** (*uri*)

Add a capability

#### **check** (*key*)

Whether specified capability is present.

**Parameter** *key* – URI or shorthand

#### **remove** (*uri*)

Remove a capability



## 2.3 content module

The `content` module provides methods for creating XML documents, parsing XML, and converting between different XML representations. It uses `ElementTree` internally.

### 2.3.1 Namespaces

The following namespace is defined in this module.

#### **BASE\_NS**

Base NETCONF namespace

Namespaces are handled just the same way as `ElementTree`. So a qualified name takes the form `{namespace}tag`. There are some utility functions for qualified names:

**qualify** (*tag*, [*ns=BASE\_NS*])

**Returns** qualified name

**unqualify** (*tag*)

**Returns** unqualified name

**Note:** It is strongly recommended to compare qualified names.

### 2.3.2 DictTree XML representation

**Note:** Where this representation is stipulated, an XML literal or `Element` is just fine as well.

`ncclient` can make use of a special syntax for XML based on Python dictionaries. It is best illustrated through an example:

```
dtree = {
    'tag': qualify('a', 'some_namespace'),
    'attrib': {'attr': 'val'},
    'subtree': [ { 'tag': 'child1' }, { 'tag': 'child2', 'text': 'some text' } ]
}
```

Calling `dtree2xml()` on `dtree` would return

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:a attr="val" xmlns:ns0="some_namespace">
  <child1 />
  <child2>some text</child2>
</ns0:a>
```

In addition to a 'pure' dictionary representation a DictTree node (including the root) may be an XML literal or an `Element` instance. The above example could thus be equivalently written as:

```
dtree2 = {
    'tag': '{ns}a',
    'attrib': {'attr': 'val'},
    'subtree': [ ET.Element('child1'), '<child2>some text</child2>' ]
}
```

### 2.3.3 Converting between different representations

Conversions *to* DictTree representation are guaranteed to be entirely dictionaries. In converting *from* DictTree representation, the argument may be any valid representation as specified.

**dtree2ele** (*spec*)

DictTree -> Element

**Return type** Element

**dtree2xml** (*spec*, [*encoding*="UTF-8"])

DictTree -> XML

**Parameter** *encoding* – character encoding

**Return type** string

**ele2dtree** (*ele*)

DictTree -> Element

**Return type** dict

**ele2xml** (*ele*)

Element -> XML

**Parameter** *encoding* – character encoding

**Return type** string

**xml2dtree** (*xml*)

XML -> DictTree

**Return type** dict

**xml2ele** (*xml*)

XML -> Element

**Return type** Element

## 2.3.4 Other utility functions

**iselement** (*obj*)

**See** `xml.etree.ElementTree.iselement()`

**find** (*ele*, *tag*, [*nslist*=, []])

If *nslist* is empty, same as `xml.etree.ElementTree.Element.find()`. If it is not, *tag* is interpreted as an unqualified name and qualified using each item in *nslist* (with a `None` item in *nslist* meaning no qualification is done). The first match is returned.

**Parameter** *nslist* – optional list of namespaces

**parse\_root** (*raw*)

Efficiently parses the root element of an XML document.

**Parameter** *raw* (string) – XML document

**Returns** a tuple of (*tag*, *attributes*), where *tag* is the (qualified) name of the element and *attributes* is a dictionary of its attributes.

**Return type** tuple

**validated\_element** (*rep*, *tag*=None, *attrs*=None, *text*=None)

Checks if the root element meets the supplied criteria. Returns a `Element` instance if so, otherwise raises `ContentError`.

**Parameters** • *tag* – tag name or a list of allowable tag names

- *attrs* – list of required attribute names, each item may be a list of allowable alternatives
- *text* – textual content to match

## 2.3.5 Errors

### exception `ContentError`

Bases: `ncclient.NCClientError`

Raised by methods of the `content` module in case of an error.

## 2.4 transport module

### 2.4.1 Base types

#### class `Session` (*capabilities*)

Base class for use by transport protocol implementations.

#### `add_listener` (*listener*)

Register a listener that will be notified of incoming messages and errors.

#### `remove_listener` (*listener*)

Unregister some listener; ignore if the listener was never registered.

#### `get_listener_instance` (*cls*)

If a listener of the specified type is registered, returns the instance.

#### `client_capabilities`

Client's Capabilities

#### `server_capabilities`

Server's Capabilities

#### `connected`

Connection status of the session.

#### `id`

A `string` representing the `session-id`. If the session has not been initialized it will be `None`

#### `can_pipeline`

Whether this session supports pipelining

#### class `SessionListener` ()

Base class for `Session` listeners, which are notified when a new NETCONF message is received or an error occurs.

**Note:** Avoid time-intensive tasks in a callback's context.

#### `callback` (*root*, *raw*)

Called when a new XML document is received. The `root` argument allows the callback to determine whether it wants to further process the document.

**Parameters**

- `root` (`tuple`) – is a tuple of (`tag`, `attributes`) where `tag` is the qualified name of the root element and `attributes` is a dictionary of its attributes (also qualified names)

- `raw` (`string`) – XML document

#### `errback` (*ex*)

Called when an error occurs.

### 2.4.2 SSH session implementation

#### static `default_unknown_host_cb` (*host*, *key*)

An unknown host callback returns `True` if it finds the key acceptable, and `False` if not.

This default callback always returns `False`, which would lead to `connect()` raising a `SSHUnknownHost` exception.

Supply another valid callback if you need to verify the host key programatically.

- Parameters**
- `host` (string) – the host for whom key needs to be verified
  - `key` (string) – a hex string representing the host key fingerprint

**class `SSHSession`** (*capabilities*)

Bases: `ncclient.transport.session.Session`

Implements a **RFC 4742** NETCONF session over SSH.

**connect** (*host*, [*port=830*, *timeout=None*, *username=None*, *password=None*, *key\_filename=None*, *allow\_agent=True*, *look\_for\_keys=True*])

Connect via SSH and initialize the NETCONF session. First attempts the publickey authentication method and then password authentication.

To disable attempting publickey authentication altogether, call with *allow\_agent* and *look\_for\_keys* as `False`. This may be needed for Cisco devices which immediately disconnect on an incorrect authentication attempt.

- Parameters**
- `host` (string) – the hostname or IP address to connect to
  - `port` (int) – by default 830, but some devices use the default SSH port of 22 so this may need to be specified
  - `timeout` (int) – an optional timeout for the TCP handshake
  - `unknown_host_cb` (see *signature*) – called when a host key is not recognized
  - `username` (string) – the username to use for SSH authentication
  - `password` (string) – the password used if using password authentication, or the passphrase to use for unlocking keys that require it
  - `key_filename` (string) – a filename where a the private key to be used can be found
  - `allow_agent` (bool) – enables querying SSH agent (if found) for keys
  - `look_for_keys` (bool) – enables looking in the usual locations for ssh keys (e.g. `~/.ssh/id_*`)

**load\_known\_hosts** (*filename=None*)

Load host keys from a `known_hosts`-style file. Can be called multiple times.

If *filename* is not specified, looks in the default locations i.e. `~/.ssh/known_hosts` and `~/ssh/known_hosts` for Windows.

**transport**

Underlying `paramiko.Transport` object. This makes it possible to call methods like `set_keepalive` on it.

## 2.4.3 Errors

**exception `TransportError`**

Bases: `ncclient.NCClientError`

**exception `SessionCloseError`**

Bases: `ncclient.transport.errors.TransportError`

**exception `SSHError`**

Bases: `ncclient.transport.errors.TransportError`

**exception `AuthenticationError`**

Bases: `ncclient.transport.errors.TransportError`

**exception `SSHUnknownHostError`**

Bases: `ncclient.transport.errors.SSHError`

## 2.5 operations module

### 2.5.1 Base types

**class** `RPC` (*session*, [*async=False*, *timeout=None*])

Base class for all operations.

Directly corresponds to `<rpc>` requests. Handles making the request, and taking delivery of the reply.

**set\_async** (*async=True*)

Set asynchronous mode for this RPC.

**set\_timeout** (*timeout*)

Set the timeout for synchronous waiting defining how long the RPC request will block on a reply before raising an error.

**reply**

`RPCReply` element if reply has been received or `None`

**error**

`Exception` type if an error occurred or `None`.

This attribute should be checked if the request was made asynchronously, so that it can be determined if `event` being set is because of a reply or error.

**Note:** This represents an error which prevented a reply from being received. An `<rpc-error>` does not fall in that category – see `RPCReply` for that.

**event**

`Event` that is set when reply has been received or error occurred.

**async**

Whether this RPC is asynchronous

**timeout**

Timeout for synchronous waiting

**id**

The *message-id* for this RPC

**session**

The `Session` object associated with this RPC

**class** `RPCReply` (*raw*)

Represents an `<rpc-reply>`. Only concerns itself with whether the operation was successful.

**Note:** If the reply has not yet been parsed there is an implicit, one-time parsing overhead to accessing the attributes defined by this class and any subclasses.

**ok**

Boolean value indicating if there were no errors.

**error**

Short for `errors` [0]; `None` if there were no errors.

**errors**

`list` of `RPCError` objects. Will be empty if there were no `<rpc-error>` elements in reply.

**class** `RPCError` (*err\_dict*)

Bases: `ncclient.operations.errors.OperationError`

Represents an `<rpc-error>`. It is an instance of `OperationError` so it can be raised like any other exception.

**type**

`string` representing *error-type* element

**severity**

`string` representing *error-severity* element

**tag**

`string` representing *error-tag* element

**path**  
     string or None; representing *error-path* element

**message**  
     string or None; representing *error-message* element

**info**  
     string or None, representing *error-info* element

## 2.5.2 NETCONF Operations

### Dependencies

Operations may have a hard dependency on some capability, or the dependency may arise at request-time due to an optional argument. In any case, a `MissingCapabilityError` is raised if the server does not support the relevant capability.

### Return type

The return type for the `request()` method depends of an operation on whether it is synchronous or asynchronous (see base class `RPC`).

- For synchronous requests, it will block waiting for the reply, and once it has been received an `RPCReply` object is returned. If an error occurred while waiting for the reply, it will be raised.
- For asynchronous requests, it will immediately return an `Event` object. This event is set when a reply is received, or an error occurs that prevents a reply from being received. The `reply` and `error` attributes can then be accessed to determine which of the two it was :-)

### General notes on parameters

#### Source / target parameters

Where an operation takes a source or target parameter, it is mainly the case that it can be a datastore name or a URL. The latter, of course, depends on the `:url` capability and whether the capability supports the specific schema of the URL. Either must be specified as a `string`.

If the source may be a `<config>` element, e.g. for `Validate`, specify in *DictTree XML representation* with the root element as `<config>`.

#### Filter parameters

Filter parameters, where applicable, can take one of the following types:

- A **tuple** of (*type*, *criteria*). Here *type* has to be one of “xpath” or “subtree”. For type “xpath”, the *criteria* should be a `string` that is a valid XPath expression. For type “subtree”, *criteria* should be in *DictTree XML representation* representing a valid subtree filter.
- A valid `<filter>` element in *DictTree XML representation*.

#### Retrieval operations

The reply object for these operations will be a `GetReply` instance.

**class Get** (*session*, *async=False*, *timeout=None*)  
     Bases: `ncclient.operations.rpc.RPC`  
     The `<get>` RPC

**request** (*filter=None*)

**Parameter** *filter* – optional; see *Filter parameters*

**Seealso** *Return type*

**class GetConfig** (*session, async=False, timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

The `<get-config>` RPC

**request** (*source, filter=None*)

**Parameters** • *source* – See *Source / target parameters*

• *filter* – optional; see *Filter parameters*

**Seealso** *Return type*

**class GetReply** (*raw*)

Bases: `ncclient.operations.rpc.RPCReply`

Adds attributes for the `<data>` element to `RPCReply`, which pertains to the `Get` and `GetConfig` operations.

**data**

Same as `data_ele`

**data\_xml**

`<data>` element as an XML string

**data\_dtree**

`<data>` element in *DictTree XML representation*

**data\_ele**

`<data>` element as an `Element`

## Locking operations

**class Lock** (*session, async=False, timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

`<lock>` RPC

**request** (*target*)

**Parameter** *target* (string) – see *Source / target parameters*

**Return type** *Return type*

**class Unlock** (*session, async=False, timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

`<unlock>` RPC

**request** (*target*)

**Parameter** *target* (string) – see *Source / target parameters*

**Return type** *Return type*

## Configuration operations

**class EditConfig** (*session, async=False, timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

`<edit-config>` RPC

**request** (*target, config, default\_operation=None, test\_option=None, error\_option=None*)

**Parameters** • *target* (string) – see *Source / target parameters*

• *config* (string or dict or `Element`) – a config element in *DictTree XML representation*

• *default\_operation* (string) – optional; one of {'merge', 'replace', 'none'}

- `test_option` (string) – optional; one of { ‘stop-on-error’, ‘continue-on-error’, ‘rollback-on-error’ }. Last option depends on the `:rollback-on-error` capability

Seealso *Return type*

**class CopyConfig** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<copy-config> RPC

**request** (*source*, *target*)

**Parameters** • *source* (string or dict or Element) – See *Source / target parameters*

- *target* (string or dict or Element) – See *Source / target parameters*

Seealso *Return type*

**class DeleteConfig** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<delete-config> RPC

**request** (*target*)

**Parameter** *target* (string or dict or Element) – See *Source / target parameters*

Seealso *Return type*

**class Validate** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<validate> RPC. Depends on the `:validate` capability.

**request** (*source*)

**Parameter** *source* (string or dict or Element) – See *Source / target parameters*

Seealso *Return type*

**class Commit** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<commit> RPC. Depends on the `:candidate` capability.

**request** (*confirmed=False*, *timeout=None*)

Requires `:confirmed-commit` capability if *confirmed* argument is `True`.

**Parameters** • *confirmed* (bool) – optional; request a confirmed commit

- *timeout* (int) – specify timeout for confirmed commit

Seealso *Return type*

**class DiscardChanges** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<discard-changes> RPC. Depends on the `:candidate` capability.

**request** (*\*args*, *\*\*kws*)

Subclasses implement this method. Here, the operation is constructed in *DictTree XML representation*, and the result of `_request()` returned.

## Session management operations

**class CloseSession** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<close-session> RPC. The connection to NETCONF server is also closed.

**class KillSession** (*session*, *async=False*, *timeout=None*)

Bases: `ncclient.operations.rpc.RPC`

<kill-session> RPC.



### Also useful

**class LockContext** (*session, target*)

A context manager for the `Lock / Unlock` pair of RPC's.

Initialise with session instance (`Session`) and lock target (*Source / target parameters*)

### 2.5.3 Errors

**exception OperationError**

Bases: `ncclient.NCClientError`

**exception TimeoutExpiredError**

Bases: `ncclient.NCClientError`

**exception MissingCapabilityError**

Bases: `ncclient.NCClientError`



# EXTENDING NCCLIENT

This is written in a 'how-to' style through code examples.

*Forthcoming*



# MODULE INDEX

## N

ncclient.capabilities, 4  
ncclient.content, 5  
ncclient.manager, 3  
ncclient.operations, 9  
ncclient.transport, 7



# INDEX

## A

add() (ncclient.capabilities.Capabilities method), 4  
add\_listener() (ncclient.transport.Session method), 7  
async (ncclient.operations.rpc.RPC attribute), 9  
AuthenticationError, 8

## B

BASE\_NS (in module ncclient.content), 5

## C

callback() (ncclient.transport.SessionListener method), 7  
can\_pipeline (ncclient.transport.Session attribute), 7  
Capabilities (class in ncclient.capabilities), 4  
CAPABILITIES (in module ncclient.capabilities), 4  
check() (ncclient.capabilities.Capabilities method), 4  
client\_capabilities (ncclient.manager.Manager attribute), 4  
client\_capabilities (ncclient.transport.Session attribute), 7  
close() (ncclient.manager.Manager method), 4  
close\_session() (ncclient.manager.Manager method), 4  
CloseSession (class in ncclient.operations), 12  
Commit (class in ncclient.operations), 12  
commit() (ncclient.manager.Manager method), 4  
connect() (in module ncclient.manager), 3  
connect() (ncclient.transport.SSHSession method), 8  
connect\_ssh() (in module ncclient.manager), 3  
connected (ncclient.manager.Manager attribute), 4  
connected (ncclient.transport.Session attribute), 7  
ContentError, 7  
copy\_config() (ncclient.manager.Manager method), 3  
CopyConfig (class in ncclient.operations), 12

## D

data (ncclient.operations.GetReply attribute), 11  
data\_dtree (ncclient.operations.GetReply attribute), 11  
data\_ele (ncclient.operations.GetReply attribute), 11  
data\_xml (ncclient.operations.GetReply attribute), 11  
default\_unknown\_host\_cb() (ncclient.transport.ssh static method), 7  
delete\_config() (ncclient.manager.Manager method), 4  
DeleteConfig (class in ncclient.operations), 12  
discard\_changes() (ncclient.manager.Manager method), 4

DiscardChanges (class in ncclient.operations), 12  
dtree2ele() (in module ncclient.content), 5  
dtree2xml() (in module ncclient.content), 6

## E

edit\_config() (ncclient.manager.Manager method), 3  
EditConfig (class in ncclient.operations), 11  
ele2dtree() (in module ncclient.content), 6  
ele2xml() (in module ncclient.content), 6  
errback() (ncclient.transport.SessionListener method), 7  
error (ncclient.operations.rpc.RPC attribute), 9  
error (ncclient.operations.rpc.RPCReply attribute), 9  
errors (ncclient.operations.rpc.RPCReply attribute), 9  
event (ncclient.operations.rpc.RPC attribute), 9

## F

find() (in module ncclient.content), 6

## G

Get (class in ncclient.operations), 10  
get() (ncclient.manager.Manager method), 3  
get\_config() (ncclient.manager.Manager method), 3  
get\_listener\_instance() (ncclient.transport.Session method), 7  
GetConfig (class in ncclient.operations), 11  
GetReply (class in ncclient.operations), 11

## I

id (ncclient.operations.rpc.RPC attribute), 9  
id (ncclient.transport.Session attribute), 7  
info (ncclient.operations.rpc.RPCError attribute), 10  
iselement() (in module ncclient.content), 6

## K

kill\_session() (ncclient.manager.Manager method), 4  
KillSession (class in ncclient.operations), 12

## L

load\_known\_hosts() (ncclient.transport.SSHSession method), 8  
Lock (class in ncclient.operations), 11  
lock() (ncclient.manager.Manager method), 4  
LockContext (class in ncclient.operations), 13  
locked() (ncclient.manager.Manager method), 4

## M

Manager (class in ncclient.manager), 3  
 message (ncclient.operations.rpc.RPCError attribute), 10  
 MissingCapabilityError, 13

## N

ncclient.capabilities (module), 4  
 ncclient.content (module), 5  
 ncclient.manager (module), 3  
 ncclient.operations (module), 9  
 ncclient.transport (module), 7

## O

ok (ncclient.operations.rpc.RPCReply attribute), 9  
 OperationError, 13

## P

parse\_root() (in module ncclient.content), 6  
 path (ncclient.operations.rpc.RPCError attribute), 9

## Q

qualify() (in module ncclient.content), 5

## R

RAISE\_ALL (in module ncclient.manager), 3  
 RAISE\_ERR (in module ncclient.manager), 3  
 RAISE\_NONE (in module ncclient.manager), 3  
 remove() (ncclient.capabilities.Capabilities method), 4  
 remove\_listener() (ncclient.transport.Session method), 7  
 reply (ncclient.operations.rpc.RPC attribute), 9  
 request() (ncclient.operations.Commit method), 12  
 request() (ncclient.operations.CopyConfig method), 12  
 request() (ncclient.operations.DeleteConfig method), 12  
 request() (ncclient.operations.DiscardChanges method), 12  
 request() (ncclient.operations.EditConfig method), 11  
 request() (ncclient.operations.Get method), 10  
 request() (ncclient.operations.GetConfig method), 11  
 request() (ncclient.operations.Lock method), 11  
 request() (ncclient.operations.Unlock method), 11  
 request() (ncclient.operations.Validate method), 12  
 RFC  
     RFC 4741, 1  
     RFC 4742, 8  
 RPC (class in ncclient.operations.rpc), 9  
 RPCError (class in ncclient.operations.rpc), 9  
 RPCReply (class in ncclient.operations.rpc), 9

## S

server\_capabilities (ncclient.manager.Manager attribute), 4  
 server\_capabilities (ncclient.transport.Session attribute), 7  
 Session (class in ncclient.transport), 7

session (ncclient.operations.rpc.RPC attribute), 9  
 session\_id (ncclient.manager.Manager attribute), 4  
 SessionCloseError, 8  
 SessionListener (class in ncclient.transport), 7  
 set\_async() (ncclient.operations.rpc.RPC method), 9  
 set\_rpc\_error\_action() (ncclient.manager.Manager method), 3  
 set\_timeout() (ncclient.operations.rpc.RPC method), 9  
 severity (ncclient.operations.rpc.RPCError attribute), 9  
 SSHError, 8  
 SSHSession (class in ncclient.transport), 8  
 SSHUnknownHostError, 8

## T

tag (ncclient.operations.rpc.RPCError attribute), 9  
 timeout (ncclient.operations.rpc.RPC attribute), 9  
 TimeoutExpiredError, 13  
 transport (ncclient.transport.SSHSession attribute), 8  
 TransportError, 8  
 type (ncclient.operations.rpc.RPCError attribute), 9

## U

Unlock (class in ncclient.operations), 11  
 unlock() (ncclient.manager.Manager method), 4  
 unqualify() (in module ncclient.content), 5

## V

Validate (class in ncclient.operations), 12  
 validate() (ncclient.manager.Manager method), 3  
 validated\_element() (in module ncclient.content), 6

## X

xml2dtree() (in module ncclient.content), 6  
 xml2ele() (in module ncclient.content), 6