
Slixmpp Documentation

Release 1.7

Nathan Fritz, Lance Stout

Jan 30, 2021

CONTENTS

1	Here's your first Slixmpp Bot:	3
2	To read if you come from SleekXMPP	5
3	Getting Started (with Examples)	9
4	Tutorials, FAQs, and How To Guides	35
5	Slixmpp Architecture and Design	55
6	API Reference	59
7	Additional Info	233
8	SleekXMPP Credits	237
	Python Module Index	239
	Index	243

Get the Code

The latest source code for Slixmpp may be found on the [Git repo](#).

```
git clone https://lab.louiz.org/poezio/slixmpp
```

An XMPP chat room is available for discussing and getting help with slixmpp.

Chat slixmpp@muc.poez.io

Reporting bugs You can report bugs at <http://lab.louiz.org/poezio/slixmpp/issues>.

Note: slixmpp is a friendly fork of [SleekXMPP](#) which goal is to use asyncio instead of threads to handle networking. See [Differences from SleekXMPP](#).

Slixmpp is an [MIT licensed](#) XMPP library for Python 3.7+,

Slixmpp's design goals and philosophy are:

Low number of dependencies Installing and using Slixmpp should be as simple as possible, without having to deal with long dependency chains.

As part of reducing the number of dependencies, some third party modules are included with Slixmpp in the `thirdparty` directory. Imports from this module first try to import an existing installed version before loading the packaged version, when possible.

Every XEP as a plugin Following Python's “batteries included” approach, the goal is to provide support for all currently active XEPs (final and draft). Since adding XEP support is done through easy to create plugins, the hope is to also provide a solid base for implementing and creating experimental XEPs.

Rewarding to work with As much as possible, Slixmpp should allow things to “just work” using sensible defaults and appropriate abstractions. XML can be ugly to work with, but it doesn't have to be that way.

HERE'S YOUR FIRST SLIXMPP BOT:

```
import asyncio
import logging

from slixmpp import ClientXMPP

class EchoBot(ClientXMPP):

    def __init__(self, jid, password):
        ClientXMPP.__init__(self, jid, password)

        self.add_event_handler("session_start", self.session_start)
        self.add_event_handler("message", self.message)

        # If you wanted more functionality, here's how to register plugins:
        # self.register_plugin('xep_0030') # Service Discovery
        # self.register_plugin('xep_0199') # XMPP Ping

        # Here's how to access plugins once you've registered them:
        # self['xep_0030'].add_feature('echo_demo')

    def session_start(self, event):
        self.send_presence()
        self.get_roster()

        # Most get_*/set_* methods from plugins use Iq stanzas, which
        # are sent asynchronously. You can almost always provide a
        # callback that will be executed when the reply is received.

    def message(self, msg):
        if msg['type'] in ('chat', 'normal'):
            msg.reply("Thanks for sending\n%(body)s" % msg).send()

if __name__ == '__main__':
    # Ideally use optparse or argparse to get JID,
    # password, and log level.

    logging.basicConfig(level=logging.DEBUG,
                        format='%(levelname)-8s %(message)s')

    xmpp = EchoBot('somejid@example.com', 'use_getpass')
    xmpp.connect()
    xmpp.process()
```


TO READ IF YOU COME FROM SLEEKXMPP

2.1 Differences from SleekXMPP

Python 3.7+ only slixmpp will work on python 3.7 and above. It may work with previous versions but we provide no guarantees.

Stanza copies The same stanza object is given through all the handlers; a handler that edits the stanza object should make its own copy.

Replies Because stanzas are not copied anymore, `Stanza.reply()` calls (for `IQs`, `Messages`, etc) now return a new object instead of editing the stanza object in-place.

Block and threaded arguments All the functions that had a `threaded=` or `block=` argument do not have it anymore. Also, `Iq.send()` **does not block anymore**.

Coroutine facilities See [Using asyncio](#)

If an event handler is a coroutine, it will be called asynchronously in the event loop instead of inside the event caller.

A CoroutineCallback class has been added to create coroutine stream handlers, which will be also handled in the event loop.

The `Iq` object's `send()` method now **always** return a Future which result will be set to the IQ reply when it is received, or to None if the IQ is not of type get or set.

Many plugins (WIP) calls which retrieve information also return the same future.

Architectural differences slixmpp does not have an event queue anymore, and instead processes handlers directly after receiving the XML stanza.

Note: If you find something that doesn't work but should, please report it.

2.2 Using asyncio

2.2.1 Block on IQ sending

`Iq.send()` now returns a Future so you can easily block with:

```
result = yield from iq.send()
```

Warning: If the reply is an IQ with an `error` type, this will raise an `IqError`, and if it timeouts, it will raise an `IqTimeout`. Don't forget to catch it.

You can still use callbacks instead.

2.2.2 XEP plugin integration

The same changes from the SleekXMPP API apply, so you can do:

```
iq_info = yield from self.xmpp['xep_0030'].get_info(jid)
```

But the following will only return a Future:

```
iq_info = self.xmpp['xep_0030'].get_info(jid)
```

2.2.3 Callbacks, Event Handlers, and Stream Handlers

IQ callbacks and *Event Handlers* can be coroutine functions; in this case, they will be scheduled in the event loop using `asyncio.async()` and not ran immediately.

A *CoroutineCallback* class has been added as well for *Stream Handlers*, which will use `asyncio.async()` to schedule the callback.

2.2.4 Running the event loop

`XMLStream.process()` is only a thin wrapper on top of `loop.run_forever()` (if `timeout` is provided then it will only run for this amount of time, and if `forever` is `False` it will run until disconnection).

Therefore you can handle the event loop in any way you like instead of using `process()`.

2.2.5 Examples

Blocking until the session is established

This code blocks until the XMPP session is fully established, which can be useful to make sure external events aren't triggering XMPP callbacks while everything is not ready.

```
import asyncio, slixmpp

client = slixmpp.ClientXMPP('jid@example', 'password')
client.connected_event = asyncio.Event()
callback = lambda _: client.connected_event.set()
client.add_event_handler('session_start', callback)
client.connect()
loop.run_until_complete(client.wait())
# do some other stuff before running the event loop, e.g.
# loop.run_until_complete(httpserver.init())
client.process()
```

Use with other asyncio-based libraries

This code interfaces with aiohttp to retrieve two pages asynchronously when the session is established, and then send the HTML content inside a simple <message>.

```
import asyncio, aiohttp, slixmpp

@asyncio.coroutine
def get_pythonorg(event):
    req = yield from aiohttp.request('get', 'http://www.python.org')
    text = yield from req.text
    client.send_message(mto='jid2@example', mbody=text)

@asyncio.coroutine
def get_asyncioorg(event):
    req = yield from aiohttp.request('get', 'http://www.asyncio.org')
    text = yield from req.text
    client.send_message(mto='jid3@example', mbody=text)

client = slixmpp.ClientXMPP('jid@example', 'password')
client.add_event_handler('session_start', get_pythonorg)
client.add_event_handler('session_start', get_asyncioorg)
client.connect()
client.process()
```

Blocking Iq

This client checks (via XEP-0092) the software used by every entity it receives a message from. After this, it sends a message to a specific JID indicating its findings.

```
import asyncio, slixmpp

class ExampleClient(slixmpp.ClientXMPP):
    def __init__(self, *args, **kwargs):
        slixmpp.ClientXMPP.__init__(self, *args, **kwargs)
        self.register_plugin('xep_0092')
        self.add_event_handler('message', self.on_message)

    @asyncio.coroutine
    def on_message(self, event):
        # You should probably handle IqError and IqTimeout exceptions here
        # but this is an example.
        version = yield from self['xep_0092'].get_version(message['from'])
        text = "%s sent me a message, he runs %s" % (message['from'],
                                                      version['software_version']['name']
                                                      )
        self.send_message(mto='master@example.tld', mbody=text)

client = ExampleClient('jid@example', 'password')
client.connect()
client.process()
```


GETTING STARTED (WITH EXAMPLES)

3.1 Slixmpp Quickstart - Echo Bot

Note: If you have any issues working through this quickstart guide join the chat room at slixmpp@muc.poez.io.

If you have not yet installed Slixmpp, do so now by either checking out a version with [Git](#).

As a basic starting project, we will create an echo bot which will reply to any messages sent to it. We will also go through adding some basic command line configuration for enabling or disabling debug log outputs and setting the username and password for the bot.

For the command line options processing, we will use the built-in `argparse` module and the `getpass` module for reading in passwords.

3.1.1 TL;DR Just Give Me the Code

As you wish: *the completed example*.

3.1.2 Overview

To get started, here is a brief outline of the structure that the final project will have:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import asyncio
import logging
import getpass

from argparse import ArgumentParser

import slixmpp

'''Here we will create our echo bot class'''

if __name__ == '__main__':
    '''Here we will configure and read command line options'''

    '''Here we will instantiate our echo bot'''
```

(continues on next page)

(continued from previous page)

```
'''Finally, we connect the bot and start listening for messages'''
```

3.1.3 Creating the EchoBot Class

There are three main types of entities within XMPP — servers, components, and clients. Since our echo bot will only be responding to a few people, and won't need to remember thousands of users, we will use a client connection. A client connection is the same type that you use with your standard IM client such as Pidgin or Psi.

Slixmpp comes with a `ClientXMPP` class which we can extend to add our message echoing feature. `ClientXMPP` requires the parameters `jid` and `password`, so we will let our `EchoBot` class accept those as well.

```
class EchoBot(slixmpp.ClientXMPP):  
  
    def __init__(self, jid, password):  
        super().__init__(jid, password)
```

Handling Session Start

The XMPP spec requires clients to broadcast its presence and retrieve its roster (buddy list) once it connects and establishes a session with the XMPP server. Until these two tasks are completed, some servers may not deliver or send messages or presence notifications to the client. So we now need to be sure that we retrieve our roster and send an initial presence once the session has started. To do that, we will register an event handler for the `session_start` event.

```
def __init__(self, jid, password):  
    super().__init__(jid, password)  
  
    self.add_event_handler('session_start', self.start)
```

Since we want the method `self.start` to execute when the `session_start` event is triggered, we also need to define the `self.start` handler.

```
async def start(self, event):  
    self.send_presence()  
    await self.get_roster()
```

Warning: Not sending an initial presence and retrieving the roster when using a client instance can prevent your program from receiving presence notifications or messages depending on the XMPP server you have chosen.

Our event handler, like every event handler, accepts a single parameter which typically is the stanza that was received that caused the event. In this case, `event` will just be an empty dictionary since there is no associated data.

Our first task of sending an initial presence is done using `send_presence`. Calling `send_presence` without any arguments will send the simplest stanza allowed in XMPP:

```
<presence />
```

The second requirement is fulfilled using `get_roster`, which will send an IQ stanza requesting the roster to the server and then wait for the response. You may be wondering what `get_roster` returns since we are not saving any return value. The roster data is saved by an internal handler to `self.roster`, and in the case of a `ClientXMPP` instance to `self.client_roster`. (The difference between `self.roster` and `self.client_roster` is

that `self.roster` supports storing roster information for multiple JIDs, which is useful for components, whereas `self.client_roster` stores roster data for just the client's JID.)

It is possible for a timeout to occur while waiting for the server to respond, which can happen if the network is excessively slow or the server is no longer responding. In that case, an `IQTimeout` is raised. Similarly, an `IQError` exception can be raised if the request contained bad data or requested the roster for the wrong user. In either case, you can wrap the `get_roster()` call in a `try/except` block to retry the roster retrieval process.

The XMPP stanzas from the roster retrieval process could look like this:

```
<iq type="get">
    <query xmlns="jabber:iq:roster" />
</iq>

<iq type="result" to="echobot@example.com" from="example.com">
    <query xmlns="jabber:iq:roster">
        <item jid="friend@example.com" subscription="both" />
    </query>
</iq>
```

Additionally, since `get_roster` is using `<iq/>` stanzas, which will always receive an answer, it should be awaited on, to keep a synchronous flow.

Responding to Messages

Now that an EchoBot instance handles `session_start`, we can begin receiving and responding to messages. Now we can register a handler for the `message` event that is raised whenever a message is received.

```
def __init__(self, jid, password):
    super().__init__(jid, password)

    self.add_event_handler('session_start', self.start)
    self.add_event_handler('message', self.message)
```

The `message` event is fired whenever a `<message />` stanza is received, including for group chat messages, errors, etc. Properly responding to messages thus requires checking the 'type' interface of the message `stanza object`. For responding to only messages addressed to our bot (and not from a chat room), we check that the type is either `normal` or `chat`. (Other potential types are `error`, `headline`, and `groupchat`.)

```
def message(self, msg):
    if msg['type'] in ('normal', 'chat'):
        msg.reply("Thanks for sending:\n%s" % msg['body']).send()
```

Let's take a closer look at the `.reply()` method used above. For message stanzas, `.reply()` accepts the parameter `body` (also as the first positional argument), which is then used as the value of the `<body />` element of the message. Setting the appropriate `to` JID is also handled by `.reply()`.

Another way to have sent the reply message would be to use `send_message`, which is a convenience method for generating and sending a message based on the values passed to it. If we were to use this method, the above code would look as so:

```
def message(self, msg):
    if msg['type'] in ('normal', 'chat'):
        self.send_message(mto=msg['from'],
                          mbody='Thanks for sending:\n%s' % msg['body'])
```

Whichever method you choose to use, the results in action will look like this:

```
<message to="echobot@example.com" from="someuser@example.net" type="chat">
    <body>Hej!</body>
</message>

<message to="someuser@example.net" type="chat">
    <body>Thanks for sending:
    Hej!</body>
</message>
```

Note: XMPP does not require stanzas sent by a client to include a `from` attribute, and leaves that responsibility to the XMPP server. However, if a sent stanza does include a `from` attribute, it must match the full JID of the client or some servers will reject it. Slixmpp thus leaves out the `from` attribute when replying using a client connection.

3.1.4 Command Line Arguments and Logging

While this isn't part of Slixmpp itself, we do want our echo bot program to be able to accept a JID and password from the command line instead of hard coding them. We will use the `argparse` module for this.

We want to accept three parameters: the JID for the echo bot, its password, and a flag for displaying the debugging logs. We also want these to be optional parameters, since passing a password directly through the command line can be a security risk.

```
if __name__ == '__main__':
    # Setup the command line arguments.
    parser = ArgumentParser(description=EchoBot.__doc__)

    # Output verbosity options.
    parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                        action="store_const", dest="loglevel",
                        const=logging.ERROR, default=logging.INFO)
    parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                        action="store_const", dest="loglevel",
                        const=logging.DEBUG, default=logging.INFO)

    # JID and password options.
    parser.add_argument("-j", "--jid", dest="jid",
                        help="JID to use")
    parser.add_argument("-p", "--password", dest="password",
                        help="password to use")

    args = parser.parse_args()

    if args.jid is None:
        args.jid = input("Username: ")
    if args.password is None:
        args.password = getpass("Password: ")
```

Since we included a flag for enabling debugging logs, we need to configure the logging module to behave accordingly.

```
if __name__ == '__main__':
    # .. option parsing from above ..
```

(continues on next page)

(continued from previous page)

```
logging.basicConfig(level=args.loglevel,
                    format='%(levelname)-8s %(message)s')
```

3.1.5 Connecting to the Server and Processing

There are three steps remaining until our echo bot is complete:

1. We need to instantiate the bot.
2. The bot needs to connect to an XMPP server.
3. We have to instruct the bot to start running and processing messages.

Creating the bot is straightforward, but we can also perform some configuration at this stage. For example, let's say we want our bot to support service discovery and pings:

```
if __name__ == '__main__':
    # ... option parsing and logging steps from above
    xmpp = EchoBot(opts.jid, opts.password)
    xmpp.register_plugin('xep_0030') # Service Discovery
    xmpp.register_plugin('xep_0199') # Ping
```

If the EchoBot class had a hard dependency on a plugin, we could register that plugin in the EchoBot.`__init__` method instead.

Now we're ready to connect and begin echoing messages. If you have the package aiodns installed, then the `slixmpp.clientxmpp.ClientXMPP.connect()` method will perform a DNS query to find the appropriate server to connect to for the given JID. If you do not have aiodns, then Slixmpp will attempt to connect to the hostname used by the JID, unless an address tuple is supplied to `slixmpp.clientxmpp.ClientXMPP.connect()`.

```
if __name__ == '__main__':
    # ... option parsing & echo bot configuration
    xmpp.connect()
    xmpp.process(forever=True)
```

The `slixmpp.basexmpp.BaseXMPP.connect()` will only schedule a connection asynchronously. To actually connect, you need to let the event loop take over. This is done with the `slixmpp.basexmpp.BaseXMPP.process()` method, which can either run forever (`forever=True`, the default), run for a (maximum) duration of time (`timeout=n`), and/or run until it gets disconnected (`forever=False`).

However, calling `process()` is not required if you already have an event loop running, so you can handle the logic around it however you like.

Note: Before slixmpp, `:meth:slixmpp.basexmpp.BaseXMPP.process` took block and threaded arguments.` These do not make sense anymore and have been removed. Slixmpp does not use threads at all.

3.1.6 The Final Product

Here then is what the final result should look like after working through the guide above. The code can also be found in the Slixmpp `examples` directory.

You can run the code using:

```
python echobot.py -d -j echobot@example.com
```

which will prompt for the password and then begin echoing messages. To test, open your regular IM client and start a chat with the echo bot. Messages you send to it should be mirrored back to you. Be careful if you are using the same JID for the echo bot that you also have logged in with another IM client. Messages could be routed to your IM client instead of the bot.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Slixmpp: The Slick XMPP Library
Copyright (C) 2010 Nathanael C. Fritz
This file is part of Slixmpp.

See the file LICENSE for copying permission.
"""

import logging
from getpass import getpass
from argparse import ArgumentParser

import slixmpp

class EchoBot(slixmpp.ClientXMPP):

    """
    A simple Slixmpp bot that will echo messages it
    receives, along with a short thank you message.
    """

    def __init__(self, jid, password):
        slixmpp.ClientXMPP.__init__(self, jid, password)

        # The session_start event will be triggered when
        # the bot establishes its connection with the server
        # and the XML streams are ready for use. We want to
        # listen for this event so that we can initialize
        # our roster.
        self.add_event_handler("session_start", self.start)

        # The message event is triggered whenever a message
        # stanza is received. Be aware that that includes
        # MUC messages and error messages.
        self.add_event_handler("message", self.message)

    @asyncio.coroutine
    def start(self, event):
        """
        Process the session_start event.
        """
```

(continues on next page)

(continued from previous page)

Typical actions for the session_start event are requesting the roster and broadcasting an initial presence stanza.

Arguments:

```
    event -- An empty dictionary. The session_start event does not provide any additional data.

"""

self.send_presence()
await self.get_roster()

def message(self, msg):
    """
    Process incoming message stanzas. Be aware that this also includes MUC messages and error messages. It is usually a good idea to check the messages's type before processing or sending replies.

    Arguments:
        msg -- The received message stanza. See the documentation for stanza objects and the Message stanza to see how it may be used.

    """
    if msg['type'] in ('chat', 'normal'):
        msg.reply("Thanks for sending\n%(body)s" % msg).send()

if __name__ == '__main__':
    # Setup the command line arguments.
    parser = ArgumentParser(description=EchoBot.__doc__)

    # Output verbosity options.
    parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                        action="store_const", dest="loglevel",
                        const=logging.ERROR, default=logging.INFO)
    parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                        action="store_const", dest="loglevel",
                        const=logging.DEBUG, default=logging.INFO)

    # JID and password options.
    parser.add_argument("-j", "--jid", dest="jid",
                        help="JID to use")
    parser.add_argument("-p", "--password", dest="password",
                        help="password to use")

    args = parser.parse_args()

    # Setup logging.
    logging.basicConfig(level=args.loglevel,
                        format='%(levelname)-8s %(message)s')

    if args.jid is None:
        args.jid = input("Username: ")
    if args.password is None:
        args.password = getpass("Password: ")
```

(continues on next page)

(continued from previous page)

```
# Setup the EchoBot and register plugins. Note that while plugins may
# have interdependencies, the order in which you register them does
# not matter.
xmpp = EchoBot(args.jid, args.password)
xmpp.register_plugin('xep_0030') # Service Discovery
xmpp.register_plugin('xep_0004') # Data Forms
xmpp.register_plugin('xep_0060') # PubSub
xmpp.register_plugin('xep_0199') # XMPP Ping

# Connect to the XMPP server and start processing XMPP stanzas.
xmpp.connect()
xmpp.process()
```

3.2 Sign in, Send a Message, and Disconnect

Note: If you have any issues working through this quickstart guide join the chat room at slixmpp@muc.poez.io.

A common use case for Slixmpp is to send one-off messages from time to time. For example, one use case could be sending out a notice when a shell script finishes a task.

We will create our one-shot bot based on the pattern explained in [Slixmpp Quickstart - Echo Bot](#). To start, we create a client class based on [ClientXMPP](#) and register a handler for the [session_start](#) event. We will also accept parameters for the JID that will receive our message, and the string content of the message.

```
import slixmpp

class SendMsgBot(slixmpp.ClientXMPP):

    def __init__(self, jid, password, recipient, msg):
        super().__init__(jid, password)

        self.recipient = recipient
        self.msg = msg

        self.add_event_handler('session_start', self.start)

    @async def start(self, event):
        self.send_presence()
        await self.get_roster()
```

Note that as in [Slixmpp Quickstart - Echo Bot](#), we need to include send an initial presence and request the roster. Next, we want to send our message, and to do that we will use [send_message](#).

```
@async def start(self, event):
    self.send_presence()
    await self.get_roster()

    self.send_message(mto=self.recipient, mbody=self.msg)
```

Finally, we need to disconnect the client using [disconnect](#). Now, sent stanzas are placed in a queue to pass them to the send routine. [disconnect](#) by default will wait for an acknowledgement from the server for at least 2.0 seconds.

This time is configurable with the `wait` parameter. If `0.0` is passed for `wait`, `disconnect` will not close the connection gracefully.

```
async def start(self, event):
    self.send_presence()
    await self.get_roster()

    self.send_message(mto=self.recipient, mbody=self.msg)

    self.disconnect()
```

Warning: If you happen to be adding stanzas to the send queue faster than the send thread can process them, then `disconnect()` will block and not disconnect.

3.2.1 Final Product

The final step is to create a small runner script for initialising our `SendMsgBot` class and adding some basic configuration options. By following the basic boilerplate pattern in [Slixmpp Quickstart - Echo Bot](#), we arrive at the code below. To experiment with this example, you can use:

```
python send_client.py -d -j oneshot@example.com -t someone@example.net -m "This is a_
→message"
```

which will prompt for the password and then log in, send your message, and then disconnect. To test, open your regular IM client with the account you wish to send messages to. When you run the `send_client.py` example and instruct it to send your IM client account a message, you should receive the message you gave. If the two JIDs you use also have a mutual presence subscription (they're on each other's buddy lists) then you will also see the `SendMsgBot` client come online and then go offline.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Slixmpp: The Slick XMPP Library
Copyright (C) 2010 Nathanael C. Fritz
This file is part of Slixmpp.

See the file LICENSE for copying permission.
"""

import logging
from getpass import getpass
from argparse import ArgumentParser

import slixmpp

class SendMsgBot(slixmpp.ClientXMPP):

    """
    A basic Slixmpp bot that will log in, send a message,
    and then log out.
    """

```

(continues on next page)

(continued from previous page)

```
def __init__(self, jid, password, recipient, message):
    slixmpp.ClientXMPP.__init__(self, jid, password)

    # The message we wish to send, and the JID that
    # will receive it.
    self.recipient = recipient
    self.msg = message

    # The session_start event will be triggered when
    # the bot establishes its connection with the server
    # and the XML streams are ready for use. We want to
    # listen for this event so that we can initialize
    # our roster.
    self.add_event_handler("session_start", self.start)

async def start(self, event):
    """
    Process the session_start event.

    Typical actions for the session_start event are
    requesting the roster and broadcasting an initial
    presence stanza.

    Arguments:
        event -- An empty dictionary. The session_start
                 event does not provide any additional
                 data.
    """
    self.send_presence()
    await self.get_roster()

    self.send_message(mto=self.recipient,
                      mbody=self.msg,
                      mtype='chat')

    self.disconnect()

if __name__ == '__main__':
    # Setup the command line arguments.
    parser = ArgumentParser(description=SendMsgBot.__doc__)

    # Output verbosity options.
    parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                        action="store_const", dest="loglevel",
                        const=logging.ERROR, default=logging.INFO)
    parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                        action="store_const", dest="loglevel",
                        const=logging.DEBUG, default=logging.INFO)

    # JID and password options.
    parser.add_argument("-j", "--jid", dest="jid",
                        help="JID to use")
    parser.add_argument("-p", "--password", dest="password",
                        help="password to use")
    parser.add_argument("-t", "--to", dest="to",
                        help="JID to send the message to")
```

(continues on next page)

(continued from previous page)

```

parser.add_argument("-m", "--message", dest="message",
                    help="message to send")

args = parser.parse_args()

# Setup logging.
logging.basicConfig(level=args.loglevel,
                    format='%(levelname)-8s %(message)s')

if args.jid is None:
    args.jid = input("Username: ")
if args.password is None:
    args.password = getpass("Password: ")
if args.to is None:
    args.to = input("Send To: ")
if args.message is None:
    args.message = input("Message: ")

# Setup the EchoBot and register plugins. Note that while plugins may
# have interdependencies, the order in which you register them does
# not matter.
xmpp = SendMsgBot(args.jid, args.password, args.to, args.message)
xmpp.register_plugin('xep_0030') # Service Discovery
xmpp.register_plugin('xep_0199') # XMPP Ping

# Connect to the XMPP server and start processing XMPP stanzas.
xmpp.connect()
xmpp.process(forever=False)

```

3.3 Create and Run a Server Component

Note: If you have any issues working through this quickstart guide join the chat room at slixmpp@muc.poez.io.

If you have not yet installed Slixmpp, do so now by either checking out a version with [Git](#).

Many XMPP applications eventually graduate to requiring to run as a server component in order to meet scalability requirements. To demonstrate how to turn an XMPP client bot into a component, we'll turn the echobot example ([Slixmpp Quickstart - Echo Bot](#)) into a component version.

The first difference is that we will add an additional import statement:

```
from slixmpp.componentxmpp import ComponentXMPP
```

Likewise, we will change the bot's class definition to match:

```
class EchoComponent(ComponentXMPP):

    def __init__(self, jid, secret, server, port):
        ComponentXMPP.__init__(self, jid, secret, server, port)
```

A component instance requires two extra parameters compared to a client instance: `server` and `port`. These specify the name and port of the XMPP server that will be accepting the component. For example, for a MUC component, the following could be used:

```
muc = ComponentXMPP('muc.slixmpp.com', '*****', 'slixmpp.com', 5555)
```

Note: The `server` value is **NOT** derived from the provided JID for the component, unlike with client connections.

One difference with the component version is that we do not have to handle the `session_start` event if we don't wish to deal with presence.

The other, main difference with components is that the '`from`' value for every stanza must be explicitly set, since components may send stanzas from multiple JIDs. To do so, the `send_message()` and `send_presence()` accept the parameters `mfrom` and `pfrom`, respectively. For any method that uses Iq stanzas, `ifrom` may be used.

3.3.1 Final Product

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Slixmpp: The Slick XMPP Library
Copyright (C) 2010 Nathanael C. Fritz
This file is part of Slixmpp.

See the file LICENSE for copying permission.
"""

import logging
from getpass import getpass
from argparse import ArgumentParser

import slixmpp
from slixmpp.componentxmpp import ComponentXMPP

class EchoComponent(ComponentXMPP):

    """
    A simple Slixmpp component that echoes messages.
    """

    def __init__(self, jid, secret, server, port):
        ComponentXMPP.__init__(self, jid, secret, server, port)

        # You don't need a session_start handler, but that is
        # where you would broadcast initial presence.

        # The message event is triggered whenever a message
        # stanza is received. Be aware that that includes
        # MUC messages and error messages.
        self.add_event_handler("message", self.message)

    def message(self, msg):
        """
        Process incoming message stanzas. Be aware that this also
        includes MUC messages and error messages. It is usually
        a good idea to check the messages's type before processing
        """


```

(continues on next page)

(continued from previous page)

or sending replies.

Since a component may send messages from any number of JIDs, it is best to always include a from JID.

Arguments:

```
msg -- The received message stanza. See the documentation
      for stanza objects and the Message stanza to see
      how it may be used.

"""
# The reply method will use the messages 'to' JID as the
# outgoing reply's 'from' JID.
msg.reply("Thanks for sending\n%(body)s" % msg).send()
```

```
if __name__ == '__main__':
    # Setup the command line arguments.
    parser = ArgumentParser(description=EchoComponent.__doc__)

    # Output verbosity options.
    parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                        action="store_const", dest="loglevel",
                        const=logging.ERROR, default=logging.INFO)
    parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                        action="store_const", dest="loglevel",
                        const=logging.DEBUG, default=logging.INFO)

    # JID and password options.
    parser.add_argument("-j", "--jid", dest="jid",
                        help="JID to use")
    parser.add_argument("-p", "--password", dest="password",
                        help="password to use")
    parser.add_argument("-s", "--server", dest="server",
                        help="server to connect to")
    parser.add_argument("-P", "--port", dest="port",
                        help="port to connect to")

    args = parser.parse_args()

    if args.jid is None:
        args.jid = input("Component JID: ")
    if args.password is None:
        args.password = getpass("Password: ")
    if args.server is None:
        args.server = input("Server: ")
    if args.port is None:
        args.port = int(input("Port: "))

    # Setup logging.
    logging.basicConfig(level=args.loglevel,
                        format='%(levelname)-8s %(message)s')

    # Setup the EchoComponent and register plugins. Note that while plugins
    # may have interdependencies, the order in which you register them does
    # not matter.
    xmpp = EchoComponent(args.jid, args.password, args.server, args.port)
    xmpp.register_plugin('xep_0030') # Service Discovery
```

(continues on next page)

(continued from previous page)

```
xmpp.register_plugin('xep_0004') # Data Forms
xmpp.register_plugin('xep_0060') # PubSub
xmpp.register_plugin('xep_0199') # XMPP Ping

# Connect to the XMPP server and start processing XMPP stanzas.
xmpp.connect()
xmpp.process()
```

3.4 Manage Presence Subscriptions

3.5 Multi-User Chat (MUC) Bot

Note: If you have any issues working through this quickstart guide join the chat room at slixmpp@muc.poez.io.

If you have not yet installed Slixmpp, do so now by either checking out a version from [Git](#).

Now that you've got the basic gist of using Slixmpp by following the echobot example ([Slixmpp Quickstart - Echo Bot](#)), we can use one of the bundled plugins to create a very popular XMPP starter project: a Multi-User Chat (MUC) bot. Our bot will login to an XMPP server, join an MUC chat room and “lurk” indefinitely, responding with a generic message to anyone that mentions its nickname. It will also greet members as they join the chat room.

3.5.1 Joining The Room

As usual, our code will be based on the pattern explained in [Slixmpp Quickstart - Echo Bot](#). To start, we create an MUCBot class based on [ClientXMPP](#) and which accepts parameters for the JID of the MUC room to join, and the nick that the bot will use inside the chat room. We also register an [event handler](#) for the [session_start](#) event.

```
import slixmpp

class MUCBot(slixmpp.ClientXMPP):

    def __init__(self, jid, password, room, nick):
        slixmpp.ClientXMPP.__init__(self, jid, password)

        self.room = room
        self.nick = nick

        self.add_event_handler("session_start", self.start)
```

After initialization, we also need to register the MUC (XEP-0045) plugin so that we can make use of the group chat plugin's methods and events.

```
xmpp.register_plugin('xep_0045')
```

Finally, we can make our bot join the chat room once an XMPP session has been established:

```
async def start(self, event):
    await self.get_roster()
    self.send_presence()
```

(continues on next page)

(continued from previous page)

```
self.plugin['xep_0045'].join_muc(self.room,
                                 self.nick)
```

Note that as in *Slixmpp Quickstart - Echo Bot*, we need to include send an initial presence and request the roster. Next, we want to join the group chat, so we call the `join_muc` method of the MUC plugin.

Note: The `plugin` attribute is dictionary that maps to instances of plugins that we have previously registered, by their names.

3.5.2 Adding Functionality

Currently, our bot just sits dormant inside the chat room, but we would like it to respond to two distinct events by issuing a generic message in each case to the chat room. In particular, when a member mentions the bot's nickname inside the chat room, and when a member joins the chat room.

Responding to Mentions

Whenever a user mentions our bot's nickname in chat, our bot will respond with a generic message resembling "*I heard that, user.*" We do this by examining all of the messages sent inside the chat and looking for the ones which contain the nickname string.

First, we register an event handler for the `groupchat_message` event inside the bot's `__init__` function.

Note: We do not register a handler for the `message` event in this bot, but if we did, the group chat message would have been sent to both handlers.

```
def __init__(self, jid, password, room, nick):
    slixmpp.ClientXMPP.__init__(self, jid, password)

    self.room = room
    self.nick = nick

    self.add_event_handler("session_start", self.start)
    self.add_event_handler("groupchat_message", self.muc_message)
```

Then, we can send our generic message whenever the bot's nickname gets mentioned.

Warning: Always check that a message is not from yourself, otherwise you will create an infinite loop responding to your own messages.

```
def muc_message(self, msg):
    if msg['mucnick'] != self.nick and self.nick in msg['body']:
        self.send_message(mto=msg['from'].bare,
                          mbody="I heard that, %s." % msg['mucnick'],
                          mtype='groupchat')
```

Greeting Members

Now we want to greet member whenever they join the group chat. To do this we will use the dynamic `muc::room@server::got_online`¹ event so it's a good idea to register an event handler for it.

Note: The `groupchat_presence` event is triggered whenever a presence stanza is received from any chat room, including any presences you send yourself. To limit event handling to a single room, use the events `muc::room@server::presence`, `muc::room@server::got_online`, or `muc::room@server::got_offline`.

```
def __init__(self, jid, password, room, nick):
    slixmpp.ClientXMPP.__init__(self, jid, password)

    self.room = room
    self.nick = nick

    self.add_event_handler("session_start", self.start)
    self.add_event_handler("groupchat_message", self.muc_message)
    self.add_event_handler("muc::%s::got_online" % self.room,
                          self.muc_online)
```

Now all that's left to do is to greet them:

```
def muc_online(self, presence):
    if presence['muc']['nick'] != self.nick:
        self.send_message(mto=presence['from'].bare,
                           mbody="Hello, %s %s" % (presence['muc']['role'],
                                                   presence['muc']['nick']),
                           mtype='groupchat')
```

3.5.3 Final Product

The final step is to create a small runner script for initialising our `MUCBot` class and adding some basic configuration options. By following the basic boilerplate pattern in [Slixmpp Quickstart - Echo Bot](#), we arrive at the code below. To experiment with this example, you can use:

```
python muc.py -d -j jid@example.com -r room@muc.example.net -n lurkbot
```

which will prompt for the password, log in, and join the group chat. To test, open your regular IM client and join the same group chat that you sent the bot to. You will see `lurkbot` as one of the members in the group chat, and that it greeted you upon entry. Send a message with the string “`lurkbot`” inside the body text, and you will also see that it responds with our pre-programmed customized message.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""

Slixmpp: The Slick XMPP Library
Copyright (C) 2010 Nathanael C. Fritz
This file is part of Slixmpp.

See the file LICENSE for copying permission.
```

(continues on next page)

¹ this is similar to the `got_online` event and is sent by the `xep_0045` plugin whenever a member joins the referenced MUC chat room.

(continued from previous page)

```

"""
import logging
from getpass import getpass
from argparse import ArgumentParser

import slixmpp

class MUCBot(slixmpp.ClientXMPP):

    """
    A simple Slixmpp bot that will greet those
    who enter the room, and acknowledge any messages
    that mentions the bot's nickname.
    """

    def __init__(self, jid, password, room, nick):
        slixmpp.ClientXMPP.__init__(self, jid, password)

        self.room = room
        self.nick = nick

        # The session_start event will be triggered when
        # the bot establishes its connection with the server
        # and the XML streams are ready for use. We want to
        # listen for this event so that we can initialize
        # our roster.
        self.add_event_handler("session_start", self.start)

        # The groupchat_message event is triggered whenever a message
        # stanza is received from any chat room. If you also also
        # register a handler for the 'message' event, MUC messages
        # will be processed by both handlers.
        self.add_event_handler("groupchat_message", self.muc_message)

        # The groupchat_presence event is triggered whenever a
        # presence stanza is received from any chat room, including
        # any presences you send yourself. To limit event handling
        # to a single room, use the events muc::room@server::presence,
        # muc::room@server::got_online, or muc::room@server::got_offline.
        self.add_event_handler("muc::%s::got_online" % self.room,
                              self.muc_online)

    @async def start(self, event):
        """
        Process the session_start event.

        Typical actions for the session_start event are
        requesting the roster and broadcasting an initial
        presence stanza.

        Arguments:
            event -- An empty dictionary. The session_start
                    event does not provide any additional
                    data.
        """

```

(continues on next page)

(continued from previous page)

```
"""
    await self.get_roster()
    self.send_presence()
    self.plugin['xep_0045'].join_muc(self.room,
                                    self.nick,
                                    # If a room password is needed, use:
                                    # password=the_room_password,
                                    )

def muc_message(self, msg):
    """
    Process incoming message stanzas from any chat room. Be aware
    that if you also have any handlers for the 'message' event,
    message stanzas may be processed by both handlers, so check
    the 'type' attribute when using a 'message' event handler.

    Whenever the bot's nickname is mentioned, respond to
    the message.

    IMPORTANT: Always check that a message is not from yourself,
    otherwise you will create an infinite loop responding
    to your own messages.

    This handler will reply to messages that mention
    the bot's nickname.

    Arguments:
        msg -- The received message stanza. See the documentation
              for stanza objects and the Message stanza to see
              how it may be used.

    """
    if msg['mucnick'] != self.nick and self.nick in msg['body']:
        self.send_message(mto=msg['from'].bare,
                          mbody="I heard that, %s." % msg['mucnick'],
                          mtype='groupchat')

def muc_online(self, presence):
    """
    Process a presence stanza from a chat room. In this case,
    presences from users that have just come online are
    handled by sending a welcome message that includes
    the user's nickname and role in the room.

    Arguments:
        presence -- The received presence stanza. See the
                   documentation for the Presence stanza
                   to see how else it may be used.

    """
    if presence['muc']['nick'] != self.nick:
        self.send_message(mto=presence['from'].bare,
                          mbody="Hello, %s %s" % (presence['muc']['role'],
                                                  presence['muc']['nick']),
                          mtype='groupchat')

if __name__ == '__main__':
    # Setup the command line arguments.
```

(continues on next page)

(continued from previous page)

```

parser = ArgumentParser()

# Output verbosity options.
parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                    action="store_const", dest="loglevel",
                    const=logging.ERROR, default=logging.INFO)
parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                    action="store_const", dest="loglevel",
                    const=logging.DEBUG, default=logging.INFO)

# JID and password options.
parser.add_argument("-j", "--jid", dest="jid",
                    help="JID to use")
parser.add_argument("-p", "--password", dest="password",
                    help="password to use")
parser.add_argument("-r", "--room", dest="room",
                    help="MUC room to join")
parser.add_argument("-n", "--nick", dest="nick",
                    help="MUC nickname")

args = parser.parse_args()

# Setup logging.
logging.basicConfig(level=args.loglevel,
                    format='%(levelname)-8s %(message)s')

if args.jid is None:
    args.jid = input("Username: ")
if args.password is None:
    args.password = getpass("Password: ")
if args.room is None:
    args.room = input("MUC room: ")
if args.nick is None:
    args.nick = input("MUC nickname: ")

# Setup the MUCBot and register plugins. Note that while plugins may
# have interdependencies, the order in which you register them does
# not matter.
xmpp = MUCBot(args.jid, args.password, args.room, args.nick)
xmpp.register_plugin('xep_0030') # Service Discovery
xmpp.register_plugin('xep_0045') # Multi-User Chat
xmpp.register_plugin('xep_0199') # XMPP Ping

# Connect to the XMPP server and start processing XMPP stanzas.
xmpp.connect()
xmpp.process()

```

3.6 Enable HTTP Proxy Support

Note: If you have any issues working through this quickstart guide join the chat room at slixmpp@muc.poez.io.

In some instances, you may wish to route XMPP traffic through an HTTP proxy, probably to get around restrictive firewalls. Slixmpp provides support for basic HTTP proxying with DIGEST authentication.

Enabling proxy support is done in two steps. The first is to instruct Slixmpp to use a proxy, and the second is to configure the proxy details:

```
xmpp = ClientXMPP('...', ...)
xmpp.use_proxy = True
xmpp.proxy_config = {
    'host': 'proxy.example.com',
    'port': 5555,
    'username': 'example_user',
    'password': '*****'
}
```

The 'username' and 'password' fields are optional if the proxy does not require authentication.

3.6.1 The Final Product

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Slixmpp: The Slick XMPP Library
Copyright (C) 2010 Nathanael C. Fritz
This file is part of Slixmpp.

See the file LICENSE for copying permission.
"""

import logging
from getpass import getpass
from argparse import ArgumentParser

import slixmpp

class EchoBot(slixmpp.ClientXMPP):

    """
    A simple Slixmpp bot that will echo messages it
    receives, along with a short thank you message.
    """

    def __init__(self, jid, password):
        slixmpp.ClientXMPP.__init__(self, jid, password)

        # The session_start event will be triggered when
        # the bot establishes its connection with the server
        # and the XML streams are ready for use. We want to
```

(continues on next page)

(continued from previous page)

```

# listen for this event so that we can initialize
# our roster.
self.add_event_handler("session_start", self.start)

# The message event is triggered whenever a message
# stanza is received. Be aware that that includes
# MUC messages and error messages.
self.add_event_handler("message", self.message)

async def start(self, event):
    """
    Process the session_start event.

    Typical actions for the session_start event are
    requesting the roster and broadcasting an initial
    presence stanza.

    Arguments:
        event -- An empty dictionary. The session_start
                event does not provide any additional
                data.

    """
    self.send_presence()
    await self.get_roster()

def message(self, msg):
    """
    Process incoming message stanzas. Be aware that this also
    includes MUC messages and error messages. It is usually
    a good idea to check the messages's type before processing
    or sending replies.

    Arguments:
        msg -- The received message stanza. See the documentation
              for stanza objects and the Message stanza to see
              how it may be used.

    """
    msg.reply("Thanks for sending\n%(body)s" % msg).send()

if __name__ == '__main__':
    # Setup the command line arguments.
    parser = ArgumentParser()

    # Output verbosity options.
    parser.add_argument("-q", "--quiet", help="set logging to ERROR",
                        action="store_const", dest="loglevel",
                        const=logging.ERROR, default=logging.INFO)
    parser.add_argument("-d", "--debug", help="set logging to DEBUG",
                        action="store_const", dest="loglevel",
                        const=logging.DEBUG, default=logging.INFO)

    # JID and password options.
    parser.add_argument("-j", "--jid", dest="jid",
                        help="JID to use")
    parser.add_argument("-p", "--password", dest="password",
                        help="password to use")

```

(continues on next page)

(continued from previous page)

```
parser.add_argument("--phost", dest="proxy_host",
                    help="Proxy hostname")
parser.add_argument("--pport", dest="proxy_port",
                    help="Proxy port")
parser.add_argument("--puser", dest="proxy_user",
                    help="Proxy username")
parser.add_argument("--ppass", dest="proxy_pass",
                    help="Proxy password")

args = parser.parse_args()

# Setup logging.
logging.basicConfig(level=args.loglevel,
                    format='%(levelname)-8s %(message)s')

if args.jid is None:
    args.jid = input("Username: ")
if args.password is None:
    args.password = getpass("Password: ")
if args.proxy_host is None:
    args.proxy_host = input("Proxy host: ")
if args.proxy_port is None:
    args.proxy_port = input("Proxy port: ")
if args.proxy_user is None:
    args.proxy_user = input("Proxy username: ")
if args.proxy_pass is None and args.proxy_user:
    args.proxy_pass = getpass("Proxy password: ")

# Setup the EchoBot and register plugins. Note that while plugins may
# have interdependencies, the order in which you register them does
# not matter.
xmpp = EchoBot(args.jid, args.password)
xmpp.register_plugin('xep_0030') # Service Discovery
xmpp.register_plugin('xep_0004') # Data Forms
xmpp.register_plugin('xep_0060') # PubSub
xmpp.register_plugin('xep_0199') # XMPP Ping

xmpp.use_proxy = True
xmpp.proxy_config = {
    'host': args.proxy_host,
    'port': int(args.proxy_port),
    'username': args.proxy_user,
    'password': args.proxy_pass}

# Connect to the XMPP server and start processing XMPP stanzas.
xmpp.connect()
xmpp.process()
```

3.7 Send a Message Every 5 Minutes

3.8 Send/Receive IQ Stanzas

Unlike Message and Presence stanzas which only use text data for basic usage, IQ stanzas require using XML payloads, and generally entail creating a new Slixmpp plugin to provide the necessary convenience methods to make working with them easier.

3.8.1 Basic Use

XMPP's use of IQ stanzas is built around namespaced `<query />` elements. For clients, just sending the empty `<query />` element will suffice for retrieving information. For example, a very basic implementation of service discovery would just need to be able to send:

```
<iq to="user@example.com" type="get" id="1">
    <query xmlns="http://jabber.org/protocol/disco#info" />
</iq>
```

Creating IQ Stanzas

Slixmpp provides built-in support for creating basic IQ stanzas this way. The relevant methods are:

- `make_iq()`
- `make_iq_get()`
- `make_iq_set()`
- `make_iq_result()`
- `make_iq_error()`
- `make_iq_query()`

These methods all follow the same pattern: create or modify an existing IQ stanza, set the '`type`' value based on the method name, and finally add a `<query />` element with the given namespace. For example, to produce the query above, you would use:

```
self.make_iq_get(queryxmlns='http://jabber.org/protocol/disco#info',
                  to='user@example.com')
```

Sending IQ Stanzas

Once an IQ stanza is created, sending it over the wire is done using its `send()` method, like any other stanza object. However, there are a few extra options to control how to wait for the query's response.

These options are:

- `block`: The default behaviour is that `send()` will block until a response is received and the response stanza will be the return value. Setting `block` to `False` will cause the call to return immediately. In which case, you will need to arrange some way to capture the response stanza if you need it.
- `timeout`: When using the blocking behaviour, the call will eventually timeout with an error. The default timeout is 30 seconds, but this may be overridden two ways. To change the timeout globally, set:

```
self.response_timeout = 10
```

To change the timeout for a single call, the `timeout` parameter works:

```
iq.send(timeout=60)
```

- `callback`: When not using a blocking call, using the `callback` argument is a simple way to register a handler that will execute whenever a response is finally received. Using this method, there is no timeout limit. In case you need to remove the callback, the name of the newly created callback is returned.

```
cb_name = iq.send(callback=self.a_callback)

# ... later if we need to cancel
self.remove_handler(cb_name)
```

Properly working with `Iq` stanzas requires handling the intended, normal flow, error responses, and timed out requests. To make this easier, two exceptions may be thrown by `send()`: `IqError` and `IqTimeout`. These exceptions only apply to the default, blocking calls.

```
try:
    resp = iq.send()
    # ... do stuff with expected Iq result
except IqError as e:
    err_resp = e.iq
    # ... handle error case
except IqTimeout:
    # ... no response received in time
    pass
```

If you do not care to distinguish between errors and timeouts, then you can combine both cases with a generic `XMPPError` exception:

```
try:
    resp = iq.send()
except XMPPError:
    # ... Don't care about the response
    pass
```

3.8.2 Advanced Use

Going beyond the basics provided by Slixmpp requires building at least a rudimentary Slixmpp plugin to create a *stanza object* for interfacing with the `Iq` payload.

See also:

- [Creating a Slixmpp Plugin](#)
- [How to Work with Stanza Objects](#)
- [Using Stream Handlers and Matchers](#)

The typical way to respond to `Iq` requests is to register stream handlers. As an example, suppose we create a stanza class named `CustomXEP` which uses the XML element `<query xmlns="custom-xep" />`, and has a `plugin_attrib` value of `custom_xep`.

There are two types of incoming `Iq` requests: `get` and `set`. You can register a handler that will accept both and then filter by type as needed, as so:

```

self.register_handler(Callback(
    'CustomXEP Handler',
    StanzaPath('iq/custom_xep'),
    self._handle_custom_iq))

# ...

def _handle_custom_iq(self, iq):
    if iq['type'] == 'get':
        # ...
        pass
    elif iq['type'] == 'set':
        # ...
        pass
    else:
        # ... This will capture error responses too
        pass

```

If you want to filter out query types beforehand, you can adjust the matching filter by using `@type=get` or `@type=set` if you are using the recommended `StanzaPath` matcher.

```

self.register_handler(Callback(
    'CustomXEP Handler',
    StanzaPath('iq@type=get/custom_xep'),
    self._handle_custom_iq_get))

# ...

def _handle_custom_iq_get(self, iq):
    assert(iq['type'] == 'get')

```


TUTORIALS, FAQS, AND HOW TO GUIDES

4.1 Supported XEPS

XEP	Description	Notes
0004	Data forms	
0009	Jabber RPC	
0012	Last Activity	
0030	Service Discovery	
0033	Extended Stanza Addressing	
0045	Multi-User Chat (MUC)	Client-side only
0050	Ad-hoc Commands	
0059	Result Set Management	
0060	Publish/Subscribe (PubSub)	Client-side only
0066	Out-of-band Data	
0078	Non-SASL Authentication	
0082	XMPP Date and Time Profiles	
0085	Chat-State Notifications	
0086	Error Condition Mappings	
0092	Software Version	
0128	Service Discovery Extensions	
0202	Entity Time	
0203	Delayed Delivery	
0224	Attention	
0249	Direct MUC Invitations	

4.2 Following XMPP: *The Definitive Guide*

Slixmpp was featured in the first edition of the O'Reilly book [XMPP: The Definitive Guide](#) by Peter Saint-Andre, Kevin Smith, and Remko Tronçon. The original source code for the book's examples can be found at <http://github.com/remko/xmpp-tdg>. An updated version of the source code, maintained to stay current with the latest Slixmpp release, is available at <http://github.com/legastero/xmpp-tdg>.

However, since publication, Slixmpp has advanced from version 0.2.1 to version 1.0 and there have been several major API changes. The most notable is the introduction of *stanza objects* which have simplified and standardized interactions with the XMPP XML stream.

What follows is a walk-through of *The Definitive Guide* highlighting the changes needed to make the code examples work with version 1.0 of Slixmpp. These changes have been kept to a minimum to preserve the correlation with the book's explanations, so be aware that some code may not use current best practices.

4.2.1 Example 2-2. (Page 26)

Implementation of a basic bot that echoes all incoming messages back to its sender.

The echo bot example requires a change to the `handleIncomingMessage` method to reflect the use of the Message *stanza object*. The "jid" field of the message object should now be "from" to match the `from` attribute of the actual XML message stanza. Likewise, "message" changes to "body" to match the `body` element of the message stanza.

Updated Code

```
def handleIncomingMessage(self, message):
    self.xmpp.send_message(message["from"], message["body"])
```

[View full source \(1\)](#) | [View original code \(1\)](#)

4.2.2 Example 14-1. (Page 215)

CheshiR IM bot implementation.

The main event handling method in the Bot class is meant to process both message events and presence update events. With the new changes in Slixmpp 1.0, extracting a CheshiR status “message” from both types of stanzas requires accessing different attributes. In the case of a message stanza, the "body" attribute would contain the CheshiR message. For a presence event, the information is stored in the "status" attribute. To handle both cases, we can test the type of the given event object and look up the proper attribute based on the type.

Like in the EchoBot example, the expression `event["jid"]` needs to change to `event["from"]` in order to get a JID object for the stanza’s sender. Because other functions in CheshiR assume that the JID is a string, the `jid` attribute is used to access the string version of the JID. A check is also added in case `user` is `None`, but the check could (and probably should) be placed in `addMessageFromUser`.

Another change is needed in `handleMessageAddedToBackend` where an HTML-IM response is created. The HTML content should be enclosed in a single element, such as a `<p>` tag.

Updated Code

```
def handleIncomingXMPPEvent(self, event):
    msgLocations = {slixmpp.stanza.presence.Presence: "status",
                    slixmpp.stanza.message.Message: "body"}

    message = event[msgLocations[type(event)]]
    user = self.backend.getUserFromJID(event["from"].jid)
    if user is not None:
        self.backend.addMessageFromUser(message, user)

def handleMessageAddedToBackend(self, message) :
    body = message.user + ": " + message.text
    htmlBody = "<p><a href='%(uri)s'>%(user)s</a>: %(message)s</p>" % {
        "uri": self.url + "/" + message.user,
        "user" : message.user, "message" : message.text }
    for subscriberJID in self.backend.getSubscriberJIDs(message.user) :
        self.xmpp.send_message(subscriberJID, body, mhtml=htmlBody)
```

[View full source \(2\)](#) | [View original code \(2\)](#)

4.2.3 Example 14-3. (Page 217)

Configurable CheshiR IM bot implementation.

Note: Since the CheshiR examples build on each other, see previous sections for corrections to code that is not marked as new in the book example.

The main difference for the configurable IM bot is the handling for the data form in handleConfigurationCommand. The test for equality with the string "1" is no longer required; Slixmpp converts boolean data form fields to the values True and False automatically.

For the method handleIncomingXMPPPresence, the attribute "jid" is again converted to "from" to get a JID object for the presence stanza's sender, and the jid attribute is used to access the string version of that JID object. A check is also added in case user is None, but the check could (and probably should) be placed in getShouldMonitorPresenceFromUser.

Updated Code

```
def handleConfigurationCommand(self, form, sessionId):
    values = form.getValues()
    monitorPresence = values["monitorPresence"]
    jid = self.xmpp.plugin["xep_0050"].sessions[sessionId]["jid"]
    user = self.backend.getUserFromJID(jid)
    self.backend.setShouldMonitorPresenceFromUser(user, monitorPresence)

def handleIncomingXMPPPresence(self, event):
    user = self.backend.getUserFromJID(event["from"].jid)
    if user is not None:
        if self.backend.getShouldMonitorPresenceFromUser(user):
            self.handleIncomingXMPPEvent(event)
```

[View full source \(3\)](#) | [View original code \(3\)](#)

4.2.4 Example 14-4. (Page 220)

CheshiR IM server component implementation.

Note: Since the CheshiR examples build on each other, see previous sections for corrections to code that is not marked as new in the book example.

Like several previous examples, a needed change is to replace subscription["from"] with subscription["from"].jid because the BaseXMPP method make_presence requires the JID to be a string.

A correction needs to be made in handleXMPPPresenceProbe because a line was left out of the original implementation; the variable user is undefined. The JID of the user can be extracted from the presence stanza's from attribute.

Since this implementation of CheshiR uses an XMPP component, it must include a from attribute in all messages that it sends. Adding the from attribute is done by including mfrom=self.xmpp.jid in calls to self.xmpp.send_message.

Updated Code

```
def handleXMPPPresenceProbe(self, event) :
    self.xmpp.send_presence(pto = event["from"])

def handleXMPPPresenceSubscription(self, subscription) :
    if subscription["type"] == "subscribe" :
        userJID = subscription["from"].jid
        self.xmpp.send_presence_subscription(pto=userJID, ptype="subscribed")
        self.xmpp.send_presence(pto = userJID)
        self.xmpp.send_presence_subscription(pto=userJID, ptype="subscribe")

def handleMessageAddedToBackend(self, message) :
    body = message.user + ":" + message.text
    for subscriberJID in self.backend.getSubscriberJIDs(message.user) :
        self.xmpp.send_message(subscriberJID, body, mfrom=self.xmpp.jid)
```

[View full source \(4\)](#) | [View original code \(4\)](#)

4.2.5 Example 14-6. (Page 223)

CheshiR IM server component with in-band registration support.

Note: Since the CheshiR examples build on each other, see previous sections for corrections to code that is not marked as new in the book example.

After applying the changes from Example 14-4 above, the registrable component implementation should work correctly.

Tip: To see how to implement in-band registration as a Slixmpp plugin, see the tutorial [tutorial-create-plugin](#).

[View full source \(5\)](#) | [View original code \(5\)](#)

4.2.6 Example 14-7. (Page 225)

Extended CheshiR IM server component implementation.

Note: Since the CheshiR examples build on each other, see previous sections for corrections to code that is not marked as new in the book example.

While the final code example can look daunting with all of the changes made, it requires very few modifications to work with the latest version of Slixmpp. Most differences are the result of CheshiR's backend functions expecting JIDs to be strings so that they can be stripped to bare JIDs. To resolve these, use the `jid` attribute of the JID objects. Also, references to `"message"` and `"jid"` attributes need to be changed to either `"body"` or `"status"`, and either `"from"` or `"to"` depending on if the object is a message or presence stanza and which of the JIDs from the stanza is needed.

Updated Code

```
def handleIncomingXMPPMessage(self, event) :
    message = self.addRecipientToMessage(event["body"], event["to"].jid)
    user = self.backend.getUserFromJID(event["from"].jid)
    self.backend.addMessageFromUser(message, user)

def handleIncomingXMPPPresence(self, event) :
    if event["to"].jid == self.componentDomain :
        user = self.backend.getUserFromJID(event["from"].jid)
        self.backend.addMessageFromUser(event["status"], user)

...

def handleXMPPPresenceSubscription(self, subscription) :
    if subscription["type"] == "subscribe" :
        userJID = subscription["from"].jid
        user = self.backend.getUserFromJID(userJID)
        contactJID = subscription["to"]
        self.xmpp.send_presence_subscription(
            pfrom=contactJID, pto=userJID, ptype="subscribed", pnick=user)
        self.sendPresenceOfContactToUser(contactJID=contactJID, userJID=userJID)
    if contactJID == self.componentDomain :
        self.sendAllContactSubscriptionRequestsToUser(userJID)
```

[View full source \(6\)](#) | [View original code \(6\)](#)

4.3 How to Work with Stanza Objects

4.3.1 Defining Stanza Interfaces

4.3.2 Creating Stanza Plugins

4.3.3 Creating a Stanza Extension

4.3.4 Overriding a Parent Stanza

4.4 Creating a Slixmpp Plugin

One of the goals of Slixmpp is to provide support for every draft or final XMPP extension ([XEP](#)). To do this, Slixmpp has a plugin mechanism for adding the functionalities required by each XEP. But even though plugins were made to quickly implement and prototype the official XMPP extensions, there is no reason you can't create your own plugin to implement your own custom XMPP-based protocol.

This guide will help walk you through the steps to implement a rudimentary version of [XEP-0077 In-band Registration](#). In-band registration was implemented in example 14-6 (page 223) of [XMPP: The Definitive Guide](#) because there was no Slixmpp plugin for XEP-0077 at the time of writing. We will partially fix that issue here by turning the example implementation from *XMPP: The Definitive Guide* into a plugin. Again, note that this will not a complete implementation, and a different, more robust, official plugin for XEP-0077 may be added to Slixmpp in the future.

Note: The example plugin created in this guide is for the server side of the registration process only. It will **NOT** be

able to register new accounts on an XMPP server.

4.4.1 First Steps

Every plugin inherits from the class `BasePlugin <slixmpp.plugins.base.BasePlugin`, and must include a `plugin_init` method. While the plugins distributed with Slixmpp must be placed in the `plugins` directory `slixmpp/plugins` to be loaded, custom plugins may be loaded from any module. To do so, use the following form when registering the plugin:

```
self.register_plugin('myplugin', module=modContaining_my_plugin)
```

The plugin name must be the same as the plugin's class name.

Now, we can open our favorite text editors and create `xep_0077.py` in `Slixmpp/slixmpp/plugins`. We want to do some basic house-keeping and declare the name and description of the XEP we are implementing. If you are creating your own custom plugin, you don't need to include the `xep` attribute.

```
"""
Creating a Slixmpp Plugin

This is a minimal implementation of XEP-0077 to serve
as a tutorial for creating Slixmpp plugins.
"""

from slixmpp.plugins.base import BasePlugin

class xep_0077(BasePlugin):
    """
    XEP-0077 In-Band Registration
    """

    def plugin_init(self):
        self.description = "In-Band Registration"
        self.xep = "0077"
```

Now that we have a basic plugin, we need to edit `slixmpp/plugins/__init__.py` to include our new plugin by adding '`xep_0077`' to the `__all__` declaration.

4.4.2 Interacting with Other Plugins

In-band registration is a feature that should be advertised through [Service Discovery](#). To do that, we tell the `xep_0030` plugin to add the "`jabber:iq:register`" feature. We put this call in a method named `post_init` which will be called once the plugin has been loaded; by doing so we advertise that we can do registrations only after we finish activating the plugin.

The `post_init` method needs to call `BasePlugin.post_init(self)` which will mark that `post_init` has been called for the plugin. Once the `Slixmpp` object begins processing, `post_init` will be called on any plugins that have not already run `post_init`. This allows you to register plugins and their dependencies without needing to worry about the order in which you do so.

Note: by adding this call we have introduced a dependency on the XEP-0030 plugin. Be sure to register '`xep_0030`' as well as '`xep_0077`'. Slixmpp does not automatically load plugin dependencies for you.

```
def post_init(self):
    BasePlugin.post_init(self)
    self.xmpp['xep_0030'].add_feature("jabber:iq:register")
```

4.4.3 Creating Custom Stanza Objects

Now, the IQ stanzas needed to implement our version of XEP-0077 are not very complex, and we could just interact with the XML objects directly just like in the *XMPP: The Definitive Guide* example. However, creating custom stanza objects is good practice.

We will create a new Registration stanza. Following the *XMPP: The Definitive Guide* example, we will add support for a username and password field. We also need two flags: registered and remove. The registered flag is sent when an already registered user attempts to register, along with their registration data. The remove flag is a request to unregister a user's account.

Adding additional fields specified in XEP-0077 will not be difficult and is left as an exercise for the reader.

Our Registration class needs to start with a few descriptions of its behaviour:

- **namespace** The namespace our stanza object lives in. In this case, "jabber:iq:register".
- **name** The name of the root XML element. In this case, the query element.
- **plugin_attrib** The name to access this type of stanza. In particular, given a registration stanza, the Registration object can be found using: iq_object['register'].
- **interfaces** A list of dictionary-like keys that can be used with the stanza object. When using "key", if there exists a method of the form getKey, setKey, or `delKey` (depending on context) then the result of calling that method will be returned. Otherwise, the value of the attribute key of the main stanza element is returned if one exists.

Note: The accessor methods currently use title case, and not camel case. Thus if you need to access an item named "methodName" you will need to use getMethodname. This naming convention might change to full camel case in a future version of Slixmpp.

- **sub_interfaces** A subset of interfaces, but these keys map to the text of any subelements that are direct children of the main stanza element. Thus, referencing iq_object['register']['username'] will either execute getUsername or return the value in the username element of the query.

If you need to access an element, say elem, that is not a direct child of the main stanza element, you will need to add getElem, setElem, and delElem. See the note above about naming conventions.

```
from slixmpp.xmlstream import ElementBase, ET, JID, register_stanza_plugin
from slixmpp import Iq

class Registration(ElementBase):
    namespace = 'jabber:iq:register'
    name = 'query'
    plugin_attrib = 'register'
    interfaces = {'username', 'password', 'registered', 'remove'}
    sub_interfaces = interfaces

    def getRegistered(self):
        present = self.xml.find('{%s}registered' % self.namespace)
        return present is not None

    def getRemove(self):
```

(continues on next page)

(continued from previous page)

```
present = self.xml.find('{%s}remove' % self.namespace)
return present is not None

def setRegistered(self, registered):
    if registered:
        self.addField('registered')
    else:
        del self['registered']

def setRemove(self, remove):
    if remove:
        self.addField('remove')
    else:
        del self['remove']

def addField(self, name):
    itemXML = ET.Element('{%s}%s' % (self.namespace, name))
    self.xml.append(itemXML)
```

Setting a `sub_interface` attribute to "" will remove that subelement. Since we want to include empty registration fields in our form, we need the `addField` method to add the empty elements.

Since the `registered` and `remove` elements are just flags, we need to add custom logic to enforce the binary behavior.

4.4.4 Extracting Stanzas from the XML Stream

Now that we have a custom stanza object, we need to be able to detect when we receive one. To do this, we register a stream handler that will pattern match stanzas off of the XML stream against our stanza object's element name and namespace. To do so, we need to create a `Callback` object which contains an XML fragment that can identify our stanza type. We can add this handler registration to our `plugin_init` method.

Also, we need to associate our `Registration` class with IQ stanzas; that requires the use of the `register_stanza_plugin` function (in `slixmpp.xmlstream.stanzabase`) which takes the class of a parent stanza type followed by the substanza type. In our case, the parent stanza is an IQ stanza, and the substanza is our registration query.

The `__handleRegistration` method referenced in the callback will be our handler function to process registration requests.

```
def plugin_init(self):
    self.description = "In-Band Registration"
    self.xep = "0077"

    self.xmpp.register_handler(
        Callback('In-Band Registration',
            MatchXPath('{%s}iq/{jabber:iq:register}query' % self.xmpp.default_ns),
            self.__handleRegistration))
    register_stanza_plugin(Iq, Registration)
```

4.4.5 Handling Incoming Stanzas and Triggering Events

There are six situations that we need to handle to finish our implementation of XEP-0077.

Registration Form Request from a New User:

```
<iq type="result">
<query xmlns="jabber:iq:register">
<username />
<password />
</query>
</iq>
```

Registration Form Request from an Existing User:

```
<iq type="result">
<query xmlns="jabber:iq:register">
<registered />
<username>Foo</username>
<password>hunter2</password>
</query>
</iq>
```

Unregister Account:

```
<iq type="result">
<query xmlns="jabber:iq:register" />
</iq>
```

Incomplete Registration:

```
<iq type="error">
<query xmlns="jabber:iq:register">
<username>Foo</username>
</query>
<error code="406" type="modify">
<not-acceptable xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
</error>
</iq>
```

Conflicting Registrations:

```
<iq type="error">
<query xmlns="jabber:iq:register">
<username>Foo</username>
<password>hunter2</password>
</query>
<error code="409" type="cancel">
<conflict xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
</error>
</iq>
```

Successful Registration:

```
<iq type="result">
<query xmlns="jabber:iq:register" />
</iq>
```

Cases 1 and 2: Registration Requests

Responding to registration requests depends on if the requesting user already has an account. If there is an account, the response should include the `registered` flag and the user's current registration information. Otherwise, we just send the fields for our registration form.

We will handle both cases by creating a `sendRegistrationForm` method that will create either an empty or full form depending on if we provide it with user data. Since we need to know which form fields to include (especially if we add support for the other fields specified in XEP-0077), we will also create a method `setForm` which will take the names of the fields we wish to include.

```
def plugin_init(self):
    self.description = "In-Band Registration"
    self.xep = "0077"
    self.form_fields = ('username', 'password')
    ... remainder of plugin_init

...

def __handleRegistration(self, iq):
    if iq['type'] == 'get':
        # Registration form requested
        userData = self.backend[iq['from'].bare]
        self.sendRegistrationForm(iq, userData)

def setForm(self, *fields):
    self.form_fields = fields

def sendRegistrationForm(self, iq, userData=None):
    reg = iq['register']
    if userData is None:
        userData = {}
    else:
        reg['registered'] = True

    for field in self.form_fields:
        data = userData.get(field, '')
        if data:
            # Add field with existing data
            reg[field] = data
        else:
            # Add a blank field
            reg.addField(field)

    iq.reply().set_payload(reg.xml)
    iq.send()
```

Note how we are able to access our Registration stanza object with `iq['register']`.

A User Backend

You might have noticed the reference to `self.backend`, which is an object that abstracts away storing and retrieving user information. Since it is not much more than a dictionary, we will leave the implementation details to the final, full source code example.

Case 3: Unregister an Account

The next simplest case to consider is responding to a request to remove an account. If we receive a `remove` flag, we instruct the backend to remove the user's account. Since your application may need to know about when users are registered or unregistered, we trigger an event using `self.xmpp.event('unregister_user', iq)`. See the component examples below for how to respond to that event.

```
def __handleRegistration(self, iq):
    if iq['type'] == 'get':
        # Registration form requested
        userData = self.backend[iq['from'].bare]
        self.sendRegistrationForm(iq, userData)
    elif iq['type'] == 'set':
        # Remove an account
        if iq['register']['remove']:
            self.backend.unregister(iq['from'].bare)
            self.xmpp.event('unregistered_user', iq)
            iq.reply().send()
    return
```

Case 4: Incomplete Registration

For the next case we need to check the user's registration to ensure it has all of the fields we wanted. The simple option that we will use is to loop over the field names and check each one; however, this means that all fields we send to the user are required. Adding optional fields is left to the reader.

Since we have received an incomplete form, we need to send an error message back to the user. We have to send a few different types of errors, so we will also create a `_sendError` method that will add the appropriate `error` element to the IQ reply.

```
def __handleRegistration(self, iq):
    if iq['type'] == 'get':
        # Registration form requested
        userData = self.backend[iq['from'].bare]
        self.sendRegistrationForm(iq, userData)
    elif iq['type'] == 'set':
        if iq['register']['remove']:
            # Remove an account
            self.backend.unregister(iq['from'].bare)
            self.xmpp.event('unregistered_user', iq)
            iq.reply().send()
        return

    for field in self.form_fields:
        if not iq['register'][field]:
            # Incomplete Registration
            self._sendError(iq, '406', 'modify', 'not-acceptable'
                           "Please fill in all fields.")
```

(continues on next page)

(continued from previous page)

```
        return

...

def _sendError(self, iq, code, error_type, name, text=''):
    iq.reply().set_payload(iq['register'].xml)
    iq.error()
    iq['error']['code'] = code
    iq['error']['type'] = error_type
    iq['error']['condition'] = name
    iq['error']['text'] = text
    iq.send()
```

Cases 5 and 6: Conflicting and Successful Registration

We are down to the final decision on if we have a successful registration. We send the user's data to the backend with the `self.backend.register` method. If it returns True, then registration has been successful. Otherwise, there has been a conflict with usernames and registration has failed. Like with unregistering an account, we trigger an event indicating that a user has been registered by using `self.xmpp.event('registered_user', iq)`. See the component examples below for how to respond to this event.

```
def __handleRegistration(self, iq):
    if iq['type'] == 'get':
        # Registration form requested
        userData = self.backend[iq['from'].bare]
        self.sendRegistrationForm(iq, userData)
    elif iq['type'] == 'set':
        if iq['register']['remove']:
            # Remove an account
            self.backend.unregister(iq['from'].bare)
            self.xmpp.event('unregistered_user', iq)
            iq.reply().send()
            return

        for field in self.form_fields:
            if not iq['register'][field]:
                # Incomplete Registration
                self._sendError(iq, '406', 'modify', 'not-acceptable',
                               "Please fill in all fields.")
        return

    if self.backend.register(iq['from'].bare, iq['register']):
        # Successful registration
        self.xmpp.event('registered_user', iq)
        iq.reply().set_payload(iq['register'].xml)
        iq.send()
    else:
        # Conflicting registration
        self._sendError(iq, '409', 'cancel', 'conflict',
                       "That username is already taken.")
```

4.4.6 Example Component Using the XEP-0077 Plugin

Alright, the moment we've been working towards - actually using our plugin to simplify our other applications. Here is a basic component that simply manages user registrations and sends the user a welcoming message when they register, and a farewell message when they delete their account.

Note that we have to register the 'xep_0030' plugin first, and that we specified the form fields we wish to use with `self.xmpp.plugin['xep_0077'].setForm('username', 'password')`.

```
import slixmpp.componentxmpp

class Example(slixmpp.componentxmpp.ComponentXMPP):

    def __init__(self, jid, password):
        slixmpp.componentxmpp.ComponentXMPP.__init__(self, jid, password, 'localhost',
→ 8888)

        self.register_plugin('xep_0030')
        self.register_plugin('xep_0077')
        self.plugin['xep_0077'].setForm('username', 'password')

        self.add_event_handler("registered_user", self.reg)
        self.add_event_handler("unregistered_user", self.unreg)

    def reg(self, iq):
        msg = "Welcome! %s" % iq['register']['username']
        self.send_message(iq['from'], msg, mfrom=self.fulljid)

    def unreg(self, iq):
        msg = "Bye! %s" % iq['register']['username']
        self.send_message(iq['from'], msg, mfrom=self.fulljid)
```

Congratulations! We now have a basic, functioning implementation of XEP-0077.

4.4.7 Complete Source Code for XEP-0077 Plugin

Here is a copy of a more complete implementation of the plugin we created, but with some additional registration fields implemented.

```
"""
Creating a Slixmpp Plugin

This is a minimal implementation of XEP-0077 to serve
as a tutorial for creating Slixmpp plugins.
"""

from slixmpp.plugins.base import BasePlugin
from slixmpp.xmlstream.handler.callback import Callback
from slixmpp.xmlstream.matcher.xpath import MatchXPath
from slixmpp.xmlstream import ElementBase, ET, JID, register_stanza_plugin
from slixmpp import Iq
import copy

class Registration(ElementBase):
    namespace = 'jabber:iq:register'
```

(continues on next page)

(continued from previous page)

```
name = 'query'
plugin_attrib = 'register'
interfaces = {'username', 'password', 'email', 'nick', 'name',
              'first', 'last', 'address', 'city', 'state', 'zip',
              'phone', 'url', 'date', 'misc', 'text', 'key',
              'registered', 'remove', 'instructions'}
sub_interfaces = interfaces

def getRegistered(self):
    present = self.xml.find('{%s}registered' % self.namespace)
    return present is not None

def getRemove(self):
    present = self.xml.find('{%s}remove' % self.namespace)
    return present is not None

def setRegistered(self, registered):
    if registered:
        self.addField('registered')
    else:
        del self['registered']

def setRemove(self, remove):
    if remove:
        self.addField('remove')
    else:
        del self['remove']

def addField(self, name):
    itemXML = ET.Element('{%s}%s' % (self.namespace, name))
    self.xml.append(itemXML)

class UserStore(object):
    def __init__(self):
        self.users = {}

    def __getitem__(self, jid):
        return self.users.get(jid, None)

    def register(self, jid, registration):
        username = registration['username']

        def filter_usernames(user):
            return user != jid and self.users[user]['username'] == username

        conflicts = filter(filter_usernames, self.users.keys())
        if conflicts:
            return False

        self.users[jid] = registration
        return True

    def unregister(self, jid):
        del self.users[jid]

class xep_0077(BasePlugin):
```

(continues on next page)

(continued from previous page)

```

"""
XEP-0077 In-Band Registration
"""

def plugin_init(self):
    self.description = "In-Band Registration"
    self.xep = "0077"
    self.form_fields = ('username', 'password')
    self.form_instructions = ""
    self.backend = UserStore()

    self.xmpp.register_handler(
        Callback('In-Band Registration',
                 MatchXPath('{%s}iq/{jabber:iq:register}query' % self.xmpp.
→default_ns),
                 self.__handleRegistration))
    register_stanza_plugin(Iq, Registration)

def post_init(self):
    BasePlugin.post_init(self)
    self.xmpp['xep_0030'].add_feature("jabber:iq:register")

def __handleRegistration(self, iq):
    if iq['type'] == 'get':
        # Registration form requested
        userData = self.backend[iq['from'].bare]
        self.sendRegistrationForm(iq, userData)
    elif iq['type'] == 'set':
        if iq['register']['remove']:
            # Remove an account
            self.backend.unregister(iq['from'].bare)
            self.xmpp.event('unregistered_user', iq)
            iq.reply().send()
            return

        for field in self.form_fields:
            if not iq['register'][field]:
                # Incomplete Registration
                self._sendError(iq, '406', 'modify', 'not-acceptable',
                               "Please fill in all fields.")
        return

    if self.backend.register(iq['from'].bare, iq['register']):
        # Successful registration
        self.xmpp.event('registered_user', iq)
        reply = iq.reply()
        reply.set_payload(iq['register'].xml)
        reply.send()
    else:
        # Conflicting registration
        self._sendError(iq, '409', 'cancel', 'conflict',
                       "That username is already taken.")

def setForm(self, *fields):
    self.form_fields = fields

def setInstructions(self, instructions):

```

(continues on next page)

(continued from previous page)

```
self.form_instructions = instructions

def sendRegistrationForm(self, iq, userData=None):
    reg = iq['register']
    if userData is None:
        userData = {}
    else:
        reg['registered'] = True

    if self.form_instructions:
        reg['instructions'] = self.form_instructions

    for field in self.form_fields:
        data = userData.get(field, '')
        if data:
            # Add field with existing data
            reg[field] = data
        else:
            # Add a blank field
            reg.addField(field)

    reply = iq.reply()
    reply.set_payload(reg.xml)
    reply.send()

def _sendError(self, iq, code, error_type, name, text=''):
    reply = iq.reply()
    reply.set_payload(iq['register'].xml)
    reply.error()
    reply['error']['code'] = code
    reply['error']['type'] = error_type
    reply['error']['condition'] = name
    reply['error']['text'] = text
    reply.send()
```

4.5 How to Use Stream Features

4.6 How SASL Authentication Works

4.7 Using Stream Handlers and Matchers

4.8 Plugin Guides

4.8.1 XEP-0030: Working with Service Discovery

XMPP networks can be composed of many individual clients, components, and servers. Determining the JIDs for these entities and the various features they may support is the role of [XEP-0030, Service Discovery](#), or “disco” for short.

Every XMPP entity may possess what are called nodes. A node is just a name for some aspect of an XMPP entity. For example, if an XMPP entity provides [Ad-Hoc Commands](#), then it will have a node named `http://jabber.`

`org/protocol/commands` which will contain information about the commands provided. Other agents using these ad-hoc commands will interact with the information provided by this node. Note that the node name is just an identifier; there is no inherent meaning.

Working with service discovery is about creating and querying these nodes. According to XEP-0030, a node may contain three types of information: identities, features, and items. (Further, extensible, information types are defined in [XEP-0128](#), but they are not yet implemented by Slixmpp.) Slixmpp provides methods to configure each of these node attributes.

Configuring Service Discovery

The design focus for the XEP-0030 plug-in is handling info and items requests in a dynamic fashion, allowing for complex policy decisions of who may receive information and how much, or use alternate backend storage mechanisms for all of the disco data. To do this, each action that the XEP-0030 plug-in performs is handed off to what is called a “node handler,” which is just a callback function. These handlers are arranged in a hierarchy that allows for a single handler to manage an entire domain of JIDs (say for a component), while allowing other handler functions to override that global behaviour for certain JIDs, or even further limited to only certain JID and node combinations.

The Dynamic Handler Hierarchy

- `global`: (JID is None, node is None)

Handlers assigned at this level for an action (such as `add_feature`) provide a global default behaviour when the action is performed.

- `jid`: (JID assigned, node is None)

At this level, handlers provide a default behaviour for actions affecting any node owned by the JID in question. This level is most useful for component connections; there is effectively no difference between this and the global level when using a client connection.

- `node`: (JID assigned, node assigned)

A handler for this level is responsible for carrying out an action for only one node, and is the most specific handler type available. These types of handlers will be most useful for “special” nodes that require special processing different than others provided by the JID, such as using access control lists, or consolidating data from other nodes.

Default Static Handlers

The XEP-0030 plug-in provides a default set of handlers that work using in-memory disco stanzas. Each handler simply performs the appropriate lookup or storage operation using these stanzas without doing any complex operations such as checking an ACL, etc.

You may find it necessary at some point to revert a particular node or JID to using the default, static handlers. To do so, use the method `restore_defaults()`. You may also elect to only convert a given set of actions instead.

Creating a Node Handler

Every node handler receives three arguments: the JID, the node, and a data parameter that will contain the relevant information for carrying out the handler's action, typically a dictionary.

The JID will always have a value, defaulting to `xmpp.boundjid.full` for components or `xmpp.boundjid.bare` for clients. The node value may be `None` or a string.

Only handlers for the actions `get_info` and `get_items` need to have return values. For these actions, `DiscoInfo` or `DiscoItems` stanzas are expected as output. It is also acceptable for handlers for these actions to generate an `XMPPError` exception when necessary.

Example Node Handler:

Here is one of the built-in default handlers as an example:

```
def add_identity(self, jid, node, data):
    """
    Add a new identity to the JID/node combination.

    The data parameter may provide:
        category -- The general category to which the agent belongs.
        itype     -- A more specific designation with the category.
        name      -- Optional human readable name for this identity.
        lang      -- Optional standard xml:lang value.
    """
    self.add_node(jid, node)
    self.nodes[(jid, node)]['info'].add_identity(
        data.get('category', ''),
        data.get('itype', ''),
        data.get('name', None),
        data.get('lang', None))
```

Adding Identities, Features, and Items

In order to maintain some backwards compatibility, the methods `add_identity`, `add_feature`, and `add_item` do not follow the method signature pattern of the other API methods (i.e. `jid`, `node`, then other options), but rather retain the parameter orders from previous plug-in versions.

Adding an Identity

Adding an identity may be done using either the older positional notation, or with keyword parameters. The example below uses the keyword arguments, but in the same order as expected using positional arguments.

```
xmpp['xep_0030'].add_identity(category='client',
                                itype='bot',
                                name='Slixmpp',
                                node='foo',
                                jid=xmpp.boundjid.full,
                                lang='no')
```

The JID and node values determine which handler will be used to perform the `add_identity` action.

The `lang` parameter allows for adding localized versions of identities using the `xml:lang` attribute.

Adding a Feature

The position ordering for `add_feature()` is to include the feature, then specify the node and then the JID. The JID and node values determine which handler will be used to perform the `add_feature` action.

```
xmpp['xep_0030'].add_feature(feature='jabber:x:data',
                               node='foo',
                               jid=xmpp.boundjid.full)
```

Adding an Item

The parameters to `add_item()` are potentially confusing due to the fact that adding an item requires two JID and node combinations: the JID and node of the item itself, and the JID and node that will own the item.

```
xmpp['xep_0030'].add_item(jid='myitemjid@example.com',
                           name='An Item!',
                           node='owner_node',
                           subnode='item_node',
                           ijid=xmpp.boundjid.full)
```

Note: In this case, the owning JID and node are provided with the parameters `ijid` and `node`.

Performing Disco Queries

The methods `get_info()` and `get_items()` are used to query remote JIDs and their nodes for disco information. Since these methods are wrappers for sending Iq stanzas, they also accept all of the parameters of the `Iq.send()` method. The `get_items()` method may also accept the boolean parameter `iterator`, which when set to `True` will return an iterator object using the [XEP-0059](#) plug-in.

```
info = yield from self['xep_0030'].get_info(jid='foo@example.com',
                                              node='bar',
                                              ifrom='baz@mycomponent.example.com',
                                              timeout=30)

items = self['xep_0030'].get_info(jid='foo@example.com',
                                   node='bar',
                                   iterator=True)
```

For more examples on how to use basic disco queries, check the `disco_browser.py` example in the `examples` directory.

Local Queries

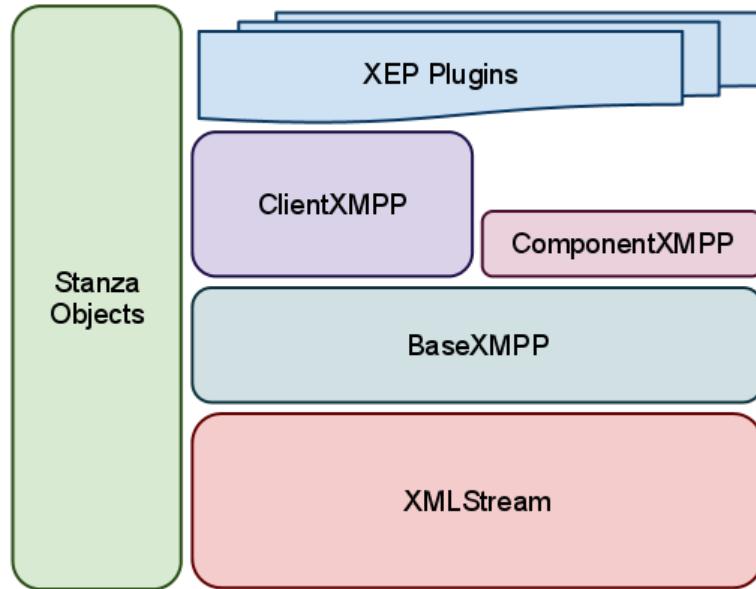
In some cases, it may be necessary to query the contents of a node owned by the client itself, or one of a component's many JIDs. The same method is used as for normal queries, with two differences. First, the parameter `local=True` must be used. Second, the return value will be a `DiscoInfo` or `DiscoItems` stanza, not a full Iq stanza.

```
info = self['xep_0030'].get_info(node='foo', local=True)
items = self['xep_0030'].get_items(jid='somejid@mycomponent.example.com',
                                   node='bar',
                                   local=True)
```

SLIXMPP ARCHITECTURE AND DESIGN

5.1 Slixmpp Architecture

The core of Slixmpp is contained in four classes: `XMLStream`, `BaseXMPP`, `ClientXMPP`, and `ComponentXMPP`. Along side this stack is a library for working with XML objects that eliminates most of the tedium of creating/manipulating XML.



5.1.1 The Foundation: `XMLStream`

`XMLStream` is a mostly XMPP-agnostic class whose purpose is to read and write from a bi-directional XML stream. It also allows for callback functions to execute when XML matching given patterns is received; these callbacks are also referred to as *stream handlers*. The class also provides a basic eventing system which can be triggered either manually or on a timed schedule.

The event loop

XMLStream instances inherit the `asyncio.BaseProtocol` class, and therefore do not have to handle reads and writes directly, but receive data through `data_received()` and write data in the socket transport.

Upon receiving data, *stream handlers* are run immediately, except if they are coroutines, in which case they are scheduled using `asyncio.async()`.

Event handlers (which are called inside *stream handlers*) work the same way.

How XML Text is Turned into Action

To demonstrate the flow of information, let's consider what happens when this bit of XML is received (with an assumed namespace of `jabber:client`):

```
<message to="user@example.com" from="friend@example.net">
    <body>Hej!</body>
</message>
```

1. Convert XML strings into objects.

Incoming text is parsed and converted into XML objects (using `ElementTree`) which are then wrapped into what are referred to as *Stanza objects*. The appropriate class for the new object is determined using a map of namespaced element names to classes.

Our incoming XML is thus turned into a *Message stanza object* because the namespaced element name `{jabber:client}message` is associated with the class `Message`.

2. Match stanza objects to callbacks.

These objects are then compared against the stored patterns associated with the registered callback handlers.

Each handler matching our *stanza object* is then added to a list.

3. Processing callbacks

Every handler in the list is then called with the *stanza object* as a parameter; if the handler is a `CoroutineCallback` then it will be scheduled in the event loop using `asyncio.async()` instead of run.

4. Raise Custom Events

Since a *stream handler* shouldn't block, if extensive processing for a stanza is required (such as needing to send and receive an `Iq` stanza), then custom events must be used. These events are not explicitly tied to the incoming XML stream and may be raised at any time.

In contrast to *stream handlers*, these functions are referred to as *event handlers*.

The code for `BaseXMPP._handle_message()` follows this pattern, and raises a 'message' event

```
self.event('message', msg)
```

5. Process Custom Events

The *event handlers* are then executed, passing the stanza as the only argument.

Note: Events may be raised without needing *stanza objects*. For example, you could use `self.event('custom', {'a': 'b'})`. You don't even need any arguments: `self.event('no_parameters')`. However, every event handler MUST accept at least one argument.

Finally, after a long trek, our message is handed off to the user's custom handler in order to do awesome stuff:

```
reply = msg.reply()  
reply['body'] = "Hey! This is awesome!"  
reply.send()
```

5.1.2 Raising XMPP Awareness: BaseXMPP

While *XMLStream* attempts to shy away from anything too XMPP specific, *BaseXMPP*'s sole purpose is to provide foundational support for sending and receiving XMPP stanzas. This support includes registering the basic message, presence, and iq stanzas, methods for creating and sending stanzas, and default handlers for incoming messages and keeping track of presence notifications.

The plugin system for adding new XEP support is also maintained by *BaseXMPP*.

5.1.3 ClientXMPP

ClientXMPP extends *BaseXMPP* with additional logic for connecting to an XMPP server by performing DNS lookups. It also adds support for stream features such as STARTTLS and SASL.

5.1.4 ComponentXMPP

ComponentXMPP is only a thin layer on top of *BaseXMPP* that implements the component handshake protocol.

5.2 Plugin Architecture

API REFERENCE

6.1 Event Index

attention

- **Data:** Message
- **Source:** [XEP_0224](#)

carbon_received

- **Data:** Message
- **Source:** [XEP_0280](#)

carbon_sent

- **Data:** Message
- **Source:** [XEP_0280](#)

changed_status

- **Data:** Presence
- **Source:** RosterItem

Triggered when a presence stanza is received from a JID with a show type different than the last presence stanza from the same JID.

changed_subscription

- **Data:** Presence
- **Source:** BaseXMPP

Triggered whenever a presence stanza with a type of subscribe, subscribed, unsubscribe, or unsubscribed is received.

Note that if the values `xmpp.auto_authorize` and `xmpp.auto_subscribe` are set to True or False, and not None, then Slixmpp will either accept or reject all subscription requests before your event handlers are called. Set these values to None if you wish to make more complex subscription decisions.

chatstate_active

- **Data:** Message
- **Source:** [xep_0085](#)

chatstate_composing

- **Data:** Message

- **Source:** xep_0085

chatstate_gone

- **Data:** Message
- **Source:** xep_0085

chatstate_inactive

- **Data:** Message
- **Source:** xep_0085

chatstate_paused

- **Data:** Message
- **Source:** xep_0085

command

- **Data:** Iq
- **Source:** [XEP_0050](#)

command_[action]

- **Data:** Iq
- **Source:** [XEP_0050](#)

connected

- **Data:** { }
- **Source:** XMLstream

Signal that a connection has been made with the XMPP server, but a session has not yet been established.

connection_failed

- **Data:** { } or Failure Stanza if available
- **Source:** XMLstream

Signal that a connection can not be established after number of attempts.

disco_info

- **Data:** DiscoInfo
- **Source:** xep_0030

Triggered whenever a disco#info result stanza is received.

disco_items

- **Data:** DiscoItems
- **Source:** xep_0030

Triggered whenever a disco#items result stanza is received.

disconnected

- **Data:** { }
- **Source:** XMLstream

Signal that the connection with the XMPP server has been lost.

entity_time

- **Data:**
- **Source:**

failed_auth

- **Data:** {}
- **Source:** ClientXMPP, xep_0078

Signal that the server has rejected the provided login credentials.

gmail_messages

- **Data:** Iq
- **Source:** gmail_notify

Signal that there are unread emails for the Gmail account associated with the current XMPP account.

gmail_notify

- **Data:** {}
- **Source:** gmail_notify

Signal that there are unread emails for the Gmail account associated with the current XMPP account.

got_offline

- **Data:** Presence
- **Source:** RosterItem

Signal that an unavailable presence stanza has been received from a JID.

got_online

- **Data:** Presence
- **Source:** RosterItem

If a presence stanza is received from a JID which was previously marked as offline, and the presence has a show type of ‘chat’, ‘dnd’, ‘away’, or ‘xa’, then this event is triggered as well.

groupchat_direct_invite

- **Data:** Message
- **Source:** direct

groupchat_invite

- **Data:** Message
- **Source:** [XEP_0045](#)

groupchat_message

- **Data:** Message
- **Source:** xep_0045

Triggered whenever a message is received from a multi-user chat room.

groupchat_presence

- **Data:** Presence

- **Source:** [xep_0045](#)

Triggered whenever a presence stanza is received from a user in a multi-user chat room.

groupchat_subject

- **Data:** Message
- **Source:** [xep_0045](#)

Triggered whenever the subject of a multi-user chat room is changed, or announced when joining a room.

ibb_stream_data

- **Data:** IBBBytestream
- **Source:** [XEP_0047](#)

ibb_stream_end

- **Data:** IBBBytestream
- **Source:** [XEP_0047](#)

ibb_stream_start

- **Data:** IBBBytestream
- **Source:** [XEP_0047](#)

jingle_message_accept

- **Data:** Message
- **Source:** [XEP_0353](#)

jingle_message_proceed

- **Data:** Message
- **Source:** [XEP_0353](#)

jingle_message_propose

- **Data:** Message
- **Source:** [XEP_0353](#)

jingle_message_reject

- **Data:** Message
- **Source:** [XEP_0353](#)

jingle_message_retract

- **Data:** Message
- **Source:** [XEP_0353](#)

killed

- **Data:**
- **Source:**

last_activity

- **Data:**
- **Source:**

marker

- **Data:** Message
- **Source:** [XEP_0333](#)

marker_acknowledged

- **Data:** Message
- **Source:** [XEP_0333](#)

marker_displayed

- **Data:** Message
- **Source:** [XEP_0333](#)

marker_received

- **Data:** Message
- **Source:** [XEP_0333](#)

message

- **Data:** Message
- **Source:** BaseXMPP

Makes the contents of message stanzas available whenever one is received. Be sure to check the message type in order to handle error messages.

message_correction

- **Data:** Message
- **Source:** [XEP_0308](#)

message_error

- **Data:** Message
- **Source:** BaseXMPP

Makes the contents of message stanzas available whenever one is received. Only handler messages with an `error` type.

message_form

- **Data:** Form
- **Source:** [xep_0004](#)

Currently the same as [message_xform](#).

message_xform

- **Data:** Form
- **Source:** [xep_0004](#)

Triggered whenever a data form is received inside a message.

muc::[room]::got_offline

- **Data:** Presence
- **Source:** [XEP_0045](#)

muc::[room]::got_online

- **Data:** Presence
- **Source:** [XEP_0045](#)

muc::[room]::message

- **Data:** Message
- **Source:** [XEP_0045](#)

muc::[room]::presence

- **Data:** Presence
- **Source:** [XEP_0045](#)

presence_available

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘available’ is received.

presence_error

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘error’ is received.

presence_form

- **Data:** Form
- **Source:** xep_0004

This event is present in the XEP-0004 plugin code, but is currently not used.

presence_probe

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘probe’ is received.

presence_subscribe

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘subscribe’ is received.

presence_subscribed

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘subscribed’ is received.

presence_unavailable

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘unavailable’ is received.

presence_unsubscribe

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘unsubscribe’ is received.

presence_unsubscribed

- **Data:** Presence
- **Source:** BaseXMPP

A presence stanza with a type of ‘unsubscribed’ is received.

pubsub_config

- **Data:** Message
- **Source:** [XEP_0060](#)

pubsub_delete

- **Data:** Message
- **Source:** [XEP_0060](#)

pubsub_publish

- **Data:** Message
- **Source:** [XEP_0060](#)

pubsub_purge

- **Data:** Message
- **Source:** [XEP_0060](#)

pubsub_retract

- **Data:** Message
- **Source:** [XEP_0060](#)

pubsub_subscription

- **Data:** Message
- **Source:** [XEP_0060](#)

reactions

- **Data:** Message
- **Source:** [XEP_0444](#)

receipt_received

- **Data:** Message
- **Source:** [XEP_0184](#)

room_activity

- **Data:** Presence
- **Source:** [XEP_0437](#)

room_activity_bare

- **Data:** Presence
- **Source:** [XEP_0437](#)

roster_update

- **Data:** Roster
- **Source:** ClientXMPP

An IQ result containing roster entries is received.

sent_presence

- **Data:** { }
- **Source:** Roster

Signal that an initial presence stanza has been written to the XML stream.

session_end

- **Data:** { }
- **Source:** XMLstream

Signal that a connection to the XMPP server has been lost and the current stream session has ended. Currently equivalent to [disconnected](#), but implementations of [XEP-0198: Stream Management](#) distinguish between the two events.

Plugins that maintain session-based state should clear themselves when this event is fired.

session_start

- **Data:** { }
- **Source:** ClientXMPP, ComponentXMPP XEP-0078

Signal that a connection to the XMPP server has been made and a session has been established.

sm_disabled

- **Data:**
- **Source:** [XEP_0198](#)

sm_enabled

- **Data:** Enabled
- **Source:** [XEP_0198](#)

socket_error

- **Data:** Socket exception object
- **Source:** XMLstream

stream:[stream id]:[peer jid]

- **Data:** IBBBytestream
- **Source:** [XEP_0047](#)

stream_error

- **Data:** StreamError
- **Source:** BaseXMPP

6.2 ClientXMPP

```
class slixmpp.clientxmpp.ClientXMPP(jid, password, plugin_config=None, plugin_whitelist=None, escape_quotes=True, sasl_mech=None, lang='en', **kwargs)
```

Slixmpp's client class. (Use only for good, not for evil.)

Typical use pattern:

```
xmpp = ClientXMPP('user@server.tld/resource', 'password')
# ... Register plugins and event handlers ...
xmpp.connect()
xmpp.process(block=False) # block=True will block the current
# thread. By default, block=False
```

Parameters

- **jid** – The JID of the XMPP user account.
- **password** – The password for the XMPP user account.
- **plugin_config** – A dictionary of plugin configurations.
- **plugin_whitelist** – A list of approved plugins that will be loaded when calling `register_plugins()`.
- **escape_quotes** – Deprecated.

connect (`address=()`, `use_ssl=False`, `force_starttls=True`, `disable_starttls=False`)

Connect to the XMPP server.

When no address is given, a SRV lookup for the server will be attempted. If that fails, the server user in the JID will be used.

Parameters

- **address** – A tuple containing the server's host and port.
- **force_starttls** – Indicates that negotiation should be aborted if the server does not advertise support for STARTTLS. Defaults to True.
- **disable_starttls** – Disables TLS for the connection. Defaults to False.
- **use_ssl** – Indicates if the older SSL connection method should be used. Defaults to False.

del_roster_item (`jid`)

Remove an item from the roster.

This is done by setting its subscription status to '`remove`'.

Parameters `jid` – The JID of the item to remove.

get_roster (`callback=None`, `timeout=None`, `timeout_callback=None`)

Request the roster from the server.

Parameters `callback` – Reference to a stream handler function. Will be executed when the roster is received.

register_feature (`name`, `handler`, `restart=False`, `order=5000`)

Register a stream feature handler.

Parameters

- **name** – The name of the stream feature.
- **handler** – The function to execute if the feature is received.
- **restart** – Indicates if feature processing should halt with this feature. Defaults to `False`.
- **order** – The relative ordering in which the feature should be negotiated. Lower values will be attempted earlier when available.

update_roster (*jid*, ***kwargs*)

Add or change a roster item.

Parameters

- **jid** – The JID of the entry to modify.
- **name** – The user's nickname for this JID.
- **subscription** – The subscription status. May be one of 'to', 'from', 'both', or 'none'. If set to 'remove', the entry will be deleted.
- **groups** – The roster groups that contain this item.
- **timeout** – The length of time (in seconds) to wait for a response before continuing if blocking is used. Defaults to `response_timeout`.
- **callback** – Optional reference to a stream handler function. Will be executed when the roster is received. Implies `block=False`.

6.3 ComponentXMPP

```
class slixmpp.componentxmpp.ComponentXMPP(jid, secret, host=None, port=None, plugin_config=None, plugin_whitelist=None, use_jc_ns=False)
```

Slixmpp's basic XMPP server component.

Use only for good, not for evil.

Parameters

- **jid** – The JID of the component.
- **secret** – The secret or password for the component.
- **host** – The server accepting the component.
- **port** – The port used to connect to the server.
- **plugin_config** – A dictionary of plugin configurations.
- **plugin_whitelist** – A list of approved plugins that will be loaded when calling `register_plugins()`.
- **use_jc_ns** – Indicates if the '`jabber:client`' namespace should be used instead of the standard '`jabber:component:accept`' namespace. Defaults to `False`.

connect (*host=None*, *port=None*, *use_ssl=False*)

Connect to the server.

Parameters

- **host** – The name of the desired server for the connection. Defaults to `server_host`.

- **port** – Port to connect to on the server. Defaults to `server_port`.
- **use_ssl** – Flag indicating if SSL should be used by connecting directly to a port using SSL.

incoming_filter (xml)

Pre-process incoming XML stanzas by converting any '`jabber:client`' namespaced elements to the component's default namespace.

Parameters `xml` – The XML stanza to pre-process.

start_stream_handler (xml)

Once the streams are established, attempt to handshake with the server to be accepted as a component.

Parameters `xml` – The incoming stream's root element.

6.4 BaseXMPP

class slixmpp.basexmpp.BaseXMPP (jid='', default_ns='jabber:client', **kwargs)

The `BaseXMPP` class adapts the generic `XMLStream` class for use with XMPP. It also provides a plugin mechanism to easily extend and add support for new XMPP features.

Parameters `default_ns` – Ensure that the correct default XML namespace is used during initialization.

Iq (*args, **kwargs)

Create an Iq stanza associated with this stream.

Message (*args, **kwargs)

Create a Message stanza associated with this stream.

Presence (*args, **kwargs)

Create a Presence stanza associated with this stream.

api

The API registry is a way to process callbacks based on JID+node combinations. Each callback in the registry is marked with:

- An API name, e.g. `xep_0030`
- The name of an action, e.g. `get_info`
- The JID that will be affected
- The node that will be affected

API handlers with no JID or node will act as global handlers, while those with a JID and no node will service all nodes for a JID, and handlers with both a JID and node will be used only for that specific combination. The handler that provides the most specificity will be used.

property auto_authorize

Auto accept or deny subscription requests.

If `True`, auto accept subscription requests. If `False`, auto deny subscription requests. If `None`, don't automatically respond.

property auto_subscribe

Auto send requests for mutual subscriptions.

If `True`, auto send mutual subscription requests.

boundjid

The JabberID (JID) used by this connection, as set after session binding. This may even be a different bare JID than what was requested.

client_roster

The single roster for the bound JID. This is the equivalent of:

```
self.roster[self.boundjid.bare]
```

exception (exception)

Process any uncaught exceptions, notably `IqError` and `IqTimeout` exceptions.

Parameters `exception` – An unhandled `Exception` object.

property fulljid

Attribute accessor for full jid

get (key, default)

Return a plugin given its name, if it has been registered.

is_component

The distinction between clients and components can be important, primarily for choosing how to handle the 'to' and 'from' JIDs of stanzas.

property jid

Attribute accessor for bare jid

make_iq (id=0, ifrom=None, ito=None, itype=None, iquery=None)

Create a new Iq stanza with a given Id and from JID.

Parameters

- `id` – An ideally unique ID value for this stanza thread. Defaults to 0.
- `ifrom` – The from JID to use for this stanza.
- `ito` – The destination JID for this stanza.
- `itype` – The Iq's type, one of: 'get', 'set', 'result', or 'error'.
- `iquery` – Optional namespace for adding a query element.

make_iq_error (id, type='cancel', condition='feature-not-implemented', text=None, ito=None, ifrom=None, iq=None)

Create an Iq stanza of type 'error'.

Parameters

- `id` – An ideally unique ID value. May use `new_id()`.
- `type` – The type of the error, such as 'cancel' or 'modify'. Defaults to 'cancel'.
- `condition` – The error condition. Defaults to 'feature-not-implemented'.
- `text` – A message describing the cause of the error.
- `ito` – The destination JID for this stanza.
- `ifrom` – The 'from' JID to use for this stanza.
- `iq` – Optionally use an existing stanza instead of generating a new one.

make_iq_get (queryxmlns=None, ito=None, ifrom=None, iq=None)

Create an Iq stanza of type 'get'.

Optionally, a query element may be added.

Parameters

- **queryxmlns** – The namespace of the query to use.
- **ito** – The destination JID for this stanza.
- **ifrom** – The 'from' JID to use for this stanza.
- **iq** – Optionally use an existing stanza instead of generating a new one.

make_iq_query (*iq=None, xmlns='', ito=None, ifrom=None*)

Create or modify an IQ stanza to use the given query namespace.

Parameters

- **iq** – Optionally use an existing stanza instead of generating a new one.
- **xmlns** – The query's namespace.
- **ito** – The destination JID for this stanza.
- **ifrom** – The 'from' JID to use for this stanza.

make_iq_result (*id=None, ito=None, ifrom=None, iq=None*)

Create an IQ stanza of type 'result' with the given ID value.

Parameters

- **id** – An ideally unique ID value. May use `new_id()`.
- **ito** – The destination JID for this stanza.
- **ifrom** – The 'from' JID to use for this stanza.
- **iq** – Optionally use an existing stanza instead of generating a new one.

make_iq_set (*sub=None, ito=None, ifrom=None, iq=None*)

Create an IQ stanza of type 'set'.

Optionally, a substanza may be given to use as the stanza's payload.

Parameters

- **sub** – Either an `ElementBase` stanza object or an `Element` XML object to use as the IQ's payload.
- **ito** – The destination JID for this stanza.
- **ifrom** – The 'from' JID to use for this stanza.
- **iq** – Optionally use an existing stanza instead of generating a new one.

make_message (*mto, mbody=None, msubject=None, mtype=None, mhtml=None, mfrom=None, mnick=None*)

Create and initialize a new Message stanza.

Parameters

- **mto** – The recipient of the message.
- **mbody** – The main contents of the message.
- **msubject** – Optional subject for the message.
- **mtype** – The message's type, such as 'chat' or 'groupchat'.
- **mhtml** – Optional HTML body content in the form of a string.
- **mfrom** – The sender of the message. If sending from a client, be aware that some servers require that the full JID of the sender be used.

- **mnick** – Optional nickname of the sender.

make_presence (*pshow=None*, *pstatus=None*, *ppriority=None*, *pto=None*, *ptype=None*,
pfrom=None, *pnick=None*)

Create and initialize a new Presence stanza.

Parameters

- **pshow** – The presence's show value.
- **pstatus** – The presence's status message.
- **ppriority** – This connection's priority.
- **pto** – The recipient of a directed presence.
- **ptype** – The type of presence, such as '`subscribe`'.
- **pfrom** – The sender of the presence.
- **pnick** – Optional nickname of the presence's sender.

make_query_roster (*iq=None*)

Create a roster query element.

Parameters iq – Optionally use an existing stanza instead of generating a new one.

max_redirects

The maximum number of consecutive see-other-host redirections that will be followed before quitting.

plugin

A dictionary mapping plugin names to plugins.

plugin_config

Configuration options for whitelisted plugins. If a plugin is registered without any configuration, and there is an entry here, it will be used.

plugin_whitelist

A list of plugins that will be loaded if `register_plugins()` is called.

process (*, forever=True, timeout=None)

Process all the available XMPP events (receiving or sending data on the socket(s), calling various registered callbacks, calling expired timers, handling signal events, etc). If timeout is None, this function will run forever. If timeout is a number, this function will return after the given time in seconds.

register_plugin (plugin, pconfig=None, module=None)

Register and configure a plugin for use in this stream.

Parameters

- **plugin** – The name of the plugin class. Plugin names must be unique.
- **pconfig** – A dictionary of configuration data for the plugin. Defaults to an empty dictionary.
- **module** – Optional reference to the module containing the plugin class if using custom plugins.

register_plugins ()

Register and initialize all built-in plugins.

Optionally, the list of plugins loaded may be limited to those contained in `plugin_whitelist`.

Plugin configurations stored in `plugin_config` will be used.

requested_jid

The JabberID (JID) requested for this connection.

property resource

Attribute accessor for jid resource

roster

The main roster object. This roster supports multiple owner JIDs, as in the case for components. For clients which only have a single JID, see [client_roster](#).

send_message (mto, mbody, msubject=None, mtype=None, mhtml=None, mfrom=None, mnick=None)

Create, initialize, and send a new Message stanza.

Parameters

- **mto** – The recipient of the message.
- **mbody** – The main contents of the message.
- **msubject** – Optional subject for the message.
- **mtype** – The message's type, such as 'chat' or 'groupchat'.
- **mhtml** – Optional HTML body content in the form of a string.
- **mfrom** – The sender of the message. If sending from a client, be aware that some servers require that the full JID of the sender be used.
- **mnick** – Optional nickname of the sender.

send_presence (pshow=None, pstatus=None, ppriority=None, pto=None, pfrom=None, ptype=None, pnick=None)

Create, initialize, and send a new Presence stanza.

Parameters

- **pshow** – The presence's show value.
- **pstatus** – The presence's status message.
- **ppriority** – This connection's priority.
- **pto** – The recipient of a directed presence.
- **ptype** – The type of presence, such as 'subscribe'.
- **pfrom** – The sender of the presence.
- **pnick** – Optional nickname of the presence's sender.

send_presence_subscription (pto, pfrom=None, ptype='subscribe', pnick=None)

Create, initialize, and send a new Presence stanza of type 'subscribe'.

Parameters

- **pto** – The recipient of a directed presence.
- **pfrom** – The sender of the presence.
- **ptype** – The type of presence, such as 'subscribe'.
- **pnick** – Optional nickname of the presence's sender.

sentpresence

Flag indicating that the initial presence broadcast has been sent. Until this happens, some servers may not behave as expected when sending stanzas.

property server

Attribute accessor for jid host

set_jid(jid)

Rip a JID apart and claim it as our own.

stanza

A reference to `slixmpp.stanza` to make accessing stanza classes easier.

start_stream_handler(xml)

Save the stream ID once the streams have been established.

Parameters `xml` – The incoming stream's root element.

stream_id

An identifier for the stream as given by the server.

use_message_ids

Messages may optionally be tagged with ID values. Setting `use_message_ids` to `True` will assign all outgoing messages an ID. Some plugin features require enabling this option.

use_origin_id

XEP-0359 `<origin-id>` tag that gets added to `<message>` stanzas.

use_presence_ids

Presence updates may optionally be tagged with ID values. Setting `use_message_ids` to `True` will assign all outgoing messages an ID.

property username

Attribute accessor for jid usernode

6.5 Exceptions

```
exception slixmpp.exceptions.XMPPError(condition='undefined-condition',           text='',
                                         etype='cancel',           extension=None,           exten-
                                         sion_ns=None, extension_args=None, clear=True)
```

A generic exception that may be raised while processing an XMPP stanza to indicate that an error response stanza should be sent.

The exception method for stanza objects extending `RootStanza` will create an error stanza and initialize any additional substanzas using the extension information included in the exception.

Meant for use in Slixmpp plugins and applications using Slixmpp.

Extension information can be included to add additional XML elements to the generated error stanza.

Parameters

- **condition** – The XMPP defined error condition. Defaults to `'undefined-condition'`.
- **text** – Human readable text describing the error.
- **etype** – The XMPP error type, such as `'cancel'` or `'modify'`. Defaults to `'cancel'`.
- **extension** – Tag name of the extension's XML content.
- **extension_ns** – XML namespace of the extensions' XML content.
- **extension_args** – Content and attributes for the extension element. Same as the additional arguments to the `Element` constructor.
- **clear** – Indicates if the stanza's contents should be removed before replying with an error. Defaults to `True`.

```
format()
    Format the error in a simple user-readable string.

exception slixmpp.exceptions.IqError(iq)
    An exception raised when an Iq stanza of type 'error' is received after making a blocking send call.

iq
    The Iq error result stanza.

exception slixmpp.exceptions.IqTimeout(iq)
    An exception which indicates that an IQ request response has not been received within the allotted time window.

iq
    The Iq stanza whose response did not arrive before the timeout expired.
```

6.6 Jabber IDs (JID)

```
class slixmpp.jid.JID(jid=None)
```

A representation of a Jabber ID, or JID.

Each JID may have three components: a user, a domain, and an optional resource. For example: `user@domain/resource`

When a resource is not used, the JID is called a bare JID. The JID is a full JID otherwise.

JID Properties:

- full** The string value of the full JID.
- jid** Alias for `full`.
- bare** The string value of the bare JID.
- node** The node portion of the JID.
- user** Alias for `node`.
- local** Alias for `node`.
- username** Alias for `node`.
- domain** The domain name portion of the JID.
- server** Alias for `domain`.
- host** Alias for `domain`.
- resource** The resource portion of the JID.

Parameters `jid(string)` – A string of the form '`[user@]domain[/resource]`'.

Raises `InvalidJID` –

unescape()

Return an unescaped JID object.

Using an unescaped JID is preferred for displaying JIDs to humans, and they should NOT be used for any other purposes than for presentation.

Returns `UnescapedJID`

New in version 1.1.10.

6.7 Stanza Objects

The `stanzabase` module provides a wrapper for the standard `ElementTree` module that makes working with XML less painful. Instead of having to manually move up and down an element tree and insert subelements and attributes, you can interact with an object that behaves like a normal dictionary or JSON object, which silently maps keys to XML attributes and elements behind the scenes.

6.7.1 Overview

The usefulness of this layer grows as the XML you have to work with becomes nested. The base unit here, `ElementBase`, can map to a single XML element, or several depending on how advanced of a mapping is desired from interface keys to XML structures. For example, a single `ElementBase` derived class could easily describe:

```
<message to="user@example.com" from="friend@example.com">
    <body>Hi!</body>
    <x:extra>
        <x:item>Custom item 1</x:item>
        <x:item>Custom item 2</x:item>
        <x:item>Custom item 3</x:item>
    </x:extra>
</message>
```

If that chunk of XML were put in the `ElementBase` instance `msg`, we could extract the data from the XML using:

```
>>> msg['extra']
['Custom item 1', 'Custom item 2', 'Custom item 3']
```

Provided we set up the handler for the '`extra`' interface to load the `<x:item>` element content into a list.

The key concept is that given an XML structure that will be repeatedly used, we can define a set of `interfaces` which when we read from, write to, or delete, will automatically manipulate the underlying XML as needed. In addition, some of these interfaces may in turn reference child objects which expose interfaces for particularly complex child elements of the original XML chunk.

See also:

[Defining Stanza Interfaces](#).

Because the `stanzabase` module was developed as part of an XMPP library, these chunks of XML are referred to as `stanzas`, and in Slixmpp we refer to a subclass of `ElementBase` which defines the interfaces needed for interacting with a given `stanza` a `stanza object`.

To make dealing with more complicated and nested `stanzas` or XML chunks easier, `stanza objects` can be composed in two ways: as iterable child objects or as plugins. Iterable child stanzas, or `substanzas`, are accessible through a special '`substanzas`' interface. This option is useful for stanzas which may contain more than one of the same kind of element. When there is only one child element, the plugin method is more useful. For plugins, a parent stanza object delegates one of its XML child elements to the plugin stanza object. Here is an example:

```
<iq type="result">
    <query xmlns="http://jabber.org/protocol/disco#info">
        <identity category="client" type="bot" name="Slixmpp Bot" />
    </query>
</iq>
```

We can arrange this stanza into two objects: an outer, wrapper object for dealing with the `<iq />` element and its attributes, and a plugin object to control the `<query />` payload element. If we give the plugin object the name '`disco_info`' (using its `ElementBase.plugin_attrib` value), then we can access the plugin as so:

```
>>> iq['disco_info']
'<query xmlns="http://jabber.org/protocol/disco#info">
<identity category="client" type="bot" name="Slixmpp Bot" />
</query>'
```

We can then drill down through the plugin object's interfaces as desired:

```
>>> iq['disco_info']['identities']
[('client', 'bot', 'Slixmpp Bot')]
```

Plugins may also add new interfaces to the parent stanza object as if they had been defined by the parent directly, and can also override the behaviour of an interface defined by the parent.

See also:

- [Creating Stanza Plugins](#)
- [Creating a Stanza Extension](#)
- [Overriding a Parent Stanza](#)

6.7.2 Registering Stanza Plugins

```
slixmpp.xmlstream.stanzabase.register_stanza_plugin(stanza, plugin, iterable=False,
                                                     overrides=False)
```

Associate a stanza object as a plugin for another stanza.

```
>>> from slixmpp.xmlstream import register_stanza_plugin
>>> register_stanza_plugin(Iq, CustomStanza)
```

Plugin stanzas marked as iterable will be included in the list of substanzas for the parent, using `parent['substanzas']`. If the attribute `plugin_multi_attrib` was defined for the plugin, then the substanza set can be filtered to only instances of the plugin class. For example, given a plugin class `Foo` with `plugin_multi_attrib = 'foos'` then:

```
parent['foos']
```

would return a collection of all `Foo` substanzas.

Parameters

- **stanza** (*class*) – The class of the parent stanza.
- **plugin** (*class*) – The class of the plugin stanza.
- **iterable** (*bool*) – Indicates if the plugin stanza should be included in the parent stanza's iterable 'substanzas' interface results.
- **overrides** (*bool*) – Indicates if the plugin should be allowed to override the interface handlers for the parent stanza, based on the plugin's `overrides` field.

New in version 1.0-Beta1: Made `register_stanza_plugin` the default name. The prior `registerStanzaPlugin` function name remains as an alias.

6.7.3 ElementBase

```
class slixmpp.xmlstream.stanzabase.ElementBase (xml=None, parent=None)
```

The core of Slixmpp's stanza XML manipulation and handling is provided by ElementBase. ElementBase wraps XML ElementTree objects and enables access to the XML contents through dictionary syntax, similar in style to the Ruby XMPP library Blather's stanza implementation.

Stanzas are defined by their name, namespace, and interfaces. For example, a simplistic Message stanza could be defined as:

```
>>> class Message(ElementBase):
...     name = "message"
...     namespace = "jabber:client"
...     interfaces = {'to', 'from', 'type', 'body'}
...     sub_interfaces = {'body'}
```

The resulting Message stanza's contents may be accessed as so:

```
>>> message['to'] = "user@example.com"
>>> message['body'] = "Hi!"
>>> message['body']
"Hi!"
>>> del message['body']
>>> message['body']
""
```

The interface values map to either custom access methods, stanza XML attributes, or (if the interface is also in sub_interfaces) the text contents of a stanza's subelement.

Custom access methods may be created by adding methods of the form “getInterface”, “setInterface”, or “delInterface”, where “Interface” is the titlecase version of the interface name.

Stanzas may be extended through the use of plugins. A plugin is simply a stanza that has a plugin_attrib value. For example:

```
>>> class MessagePlugin(ElementBase):
...     name = "custom_plugin"
...     namespace = "custom"
...     interfaces = {'useful_thing', 'custom'}
...     plugin_attrib = "custom"
```

The plugin stanza class must be associated with its intended container stanza by using register_stanza_plugin as so:

```
>>> register_stanza_plugin(Message, MessagePlugin)
```

The plugin may then be accessed as if it were built-in to the parent stanza:

```
>>> message['custom']['useful_thing'] = 'foo'
```

If a plugin provides an interface that is the same as the plugin's plugin_attrib value, then the plugin's interface may be assigned directly from the parent stanza, as shown below, but retrieving information will require all interfaces to be used, as so:

```
>>> # Same as using message['custom']['custom']
>>> message['custom'] = 'bar'
>>> # Must use all interfaces
```

(continues on next page)

(continued from previous page)

```
>>> message['custom']['custom']
'bar'
```

If the plugin sets `is_extension` to True, then both setting and getting an interface value that is the same as the plugin's `plugin_attrib` value will work, as so:

```
>>> message['custom'] = 'bar' # Using is_extension=True
>>> message['custom']
'bar'
```

Parameters

- `xml` – Initialize the stanza object with an existing XML object.
- `parent` – Optionally specify a parent stanza object will contain this substanza.

`__bool__()`

Stanza objects should be treated as True in boolean contexts.

`__copy__()`

Return a copy of the stanza object that does not share the same underlying XML object.

`__delitem__(attrib)`

Delete the value of a stanza interface using dict-like syntax.

Example:

```
>>> msg['body'] = "Hi!"
>>> msg['body']
'Hi!'
>>> del msg['body']
>>> msg['body']
''
```

Stanza interfaces are typically mapped directly to the underlying XML object, but can be overridden by the presence of a `del_attrib` method (or `del_foo` where the interface is named '`foo`', etc).

The effect of deleting a stanza interface value named `foo` will be one of:

1. Call `del_foo` override handler, if it exists.
2. Call `del_foo`, if it exists.
3. Call `delFoo`, if it exists.
4. Delete `foo` element, if '`foo`' is in `sub_interfaces`.
5. Remove `foo` element if '`foo`' is in `bool_interfaces`.
6. Delete top level XML attribute named `foo`.
7. Remove the `foo` plugin, if it was loaded.
8. Do nothing.

Parameters `attrib` – The name of the affected stanza interface.

`__eq__(other)`

Compare the stanza object with another to test for equality.

Stanzas are equal if their interfaces return the same values, and if they are both instances of `ElementBase`.

Parameters `other` (`ElementBase`) – The stanza object to compare against.

`__getitem__(full_attrib)`

Return the value of a stanza interface using dict-like syntax.

Example:

```
>>> msg['body']
'Message contents'
```

Stanza interfaces are typically mapped directly to the underlying XML object, but can be overridden by the presence of a `get_attrib` method (or `get_foo` where the interface is named '`foo`', etc).

The search order for interface value retrieval for an interface named '`foo`' is:

1. The list of substanzas ('`substanzas`')
2. The result of calling the `get_foo` override handler.
3. The result of calling `get_foo`.
4. The result of calling `getFoo`.
5. The contents of the `foo` subelement, if `foo` is listed in `sub_interfaces`.
6. True or False depending on the existence of a `foo` subelement and `foo` is in `bool_interfaces`.
7. The value of the `foo` attribute of the XML object.
8. The plugin named '`foo`'
9. An empty string.

Parameters `full_attrib` (`string`) – The name of the requested stanza interface.

`__init__(xml=None, parent=None)`

Initialize self. See `help(type(self))` for accurate signature.

`__iter__()`

Return an iterator object for the stanza's substanzas.

The iterator is the stanza object itself. Attempting to use two iterators on the same stanza at the same time is discouraged.

`__len__()`

Return the number of iterable substanzas in this stanza.

`__ne__(other)`

Compare the stanza object with another to test for inequality.

Stanzas are not equal if their interfaces return different values, or if they are not both instances of `ElementBase`.

Parameters `other` (`ElementBase`) – The stanza object to compare against.

`__next__()`

Return the next iterable substanza.

`__repr__()`

Use the stanza's serialized XML as its representation.

`__setitem__(attrib, value)`

Set the value of a stanza interface using dictionary-like syntax.

Example:

```
>>> msg['body'] = "Hi!"
>>> msg['body']
'Hi!'
```

Stanza interfaces are typically mapped directly to the underlying XML object, but can be overridden by the presence of a `set_attrib` method (or `set_foo` where the interface is named '`foo`', etc).

The effect of interface value assignment for an interface named '`foo`' will be one of:

1. Delete the interface's contents if the value is `None`.
2. Call the `set_foo` override handler, if it exists.
3. Call `set_foo`, if it exists.
4. Call `setFoo`, if it exists.
5. Set the text of a `foo` element, if '`foo`' is in `sub_interfaces`.
6. Add or remove an empty subelement `foo` if `foo` is in `bool_interfaces`.
7. Set the value of a top level XML attribute named `foo`.
8. Attempt to pass the value to a plugin named '`foo`' using the plugin's '`foo`' interface.
9. Do nothing.

Parameters

- **attrib** (*string*) – The name of the stanza interface to modify.
- **value** – The new value of the stanza interface.

__str__(top_level_ns=True)

Return a string serialization of the underlying XML object.

See also:

[XML Serialization](#)

Parameters `top_level_ns` (*bool*) – Display the top-most namespace. Defaults to True.

__weakref__

list of weak references to the object (if defined)

_del_attr(name)

Remove a top level attribute of the XML object.

Parameters `name` – The name of the attribute.

_del_sub(name, all=False, lang=None)

Remove sub elements that match the given name or XPath.

If the element is in a path, then any parent elements that become empty after deleting the element may also be deleted if requested by setting `all=True`.

Parameters

- **name** – The name or XPath expression for the element(s) to remove.
- **all** (*bool*) – If True, remove all empty elements in the path to the deleted element. Defaults to False.

`_get_attr (name, default="")`

Return the value of a top level attribute of the XML object.

In case the attribute has not been set, a default value can be returned instead. An empty string is returned if no other default is supplied.

Parameters

- **name** – The name of the attribute.
- **default** – Optional value to return if the attribute has not been set. An empty string is returned otherwise.

`_get_stanza_values ()`

Return A JSON/dictionary version of the XML content exposed through the stanza's interfaces:

```
>>> msg = Message()  
>>> msg.values  
{'body': '', 'from': '', 'mucnick': '', 'mucroom': '',  
'to': '', 'type': 'normal', 'id': '', 'subject': ''}
```

Likewise, assigning to `values` will change the XML content:

```
>>> msg = Message()  
>>> msg.values = {'body': 'Hi!', 'to': 'user@example.com'}  
>>> msg  
'<message to="user@example.com"><body>Hi!</body></message>'
```

New in version 1.0-Beta1.

`_get_sub_text (name, default="", lang=None)`

Return the text contents of a sub element.

In case the element does not exist, or it has no textual content, a default value can be returned instead. An empty string is returned if no other default is supplied.

Parameters

- **name** – The name or XPath expression of the element.
- **default** – Optional default to return if the element does not exists. An empty string is returned otherwise.

`_set_attr (name, value)`

Set the value of a top level attribute of the XML object.

If the new value is None or an empty string, then the attribute will be removed.

Parameters

- **name** – The name of the attribute.
- **value** – The new value of the attribute, or None or '' to remove it.

`_set_stanza_values (values)`

Set multiple stanza interface values using a dictionary.

Stanza plugin values may be set using nested dictionaries.

Parameters **values** – A dictionary mapping stanza interface with values. Plugin interfaces may accept a nested dictionary that will be used recursively.

New in version 1.0-Beta1.

_set_sub_text (*name*, *text=None*, *keep=False*, *lang=None*)

Set the text contents of a sub element.

In case the element does not exist, a element will be created, and its text contents will be set.

If the text is set to an empty string, or None, then the element will be removed, unless keep is set to True.

Parameters

- **name** – The name or XPath expression of the element.
- **text** – The new textual content of the element. If the text is an empty string or None, the element will be removed unless the parameter keep is True.
- **keep** – Indicates if the element should be kept if its text is removed. Defaults to False.

append (*item*)

Append either an XML object or a stanza to this stanza object.

If a stanza object is appended, it will be added to the list of iterable stanzas.

Allows stanza objects to be used like lists.

Parameters **item** – Either an XML object or a stanza object to add to this stanza's contents.

appendxml (*xml*)

Append an XML object to the stanza's XML.

The added XML will not be included in the list of iterable substanzas.

Parameters **xml** (*XML*) – The XML object to add to the stanza.

bool_interfaces = {}

A subset of *interfaces* which maps the presence of subelements to boolean values. Using this set allows for quickly checking for the existence of empty subelements like <required />.

New in version 1.1.

clear()

Remove all XML element contents and plugins.

Any attribute values will be preserved.

enable (*attrib*, *lang=None*)

Enable and initialize a stanza plugin.

Alias for *init_plugin()*.

Parameters **attrib** (*string*) – The *plugin_attrib* value of the plugin to enable.

get (*key*, *default=None*)

Return the value of a stanza interface.

If the found value is None or an empty string, return the supplied default value.

Allows stanza objects to be used like dictionaries.

Parameters

- **key** (*string*) – The name of the stanza interface to check.
- **default** – Value to return if the stanza interface has a value of None or " ". Will default to returning None.

get_stanza_values()

Return A JSON/dictionary version of the XML content exposed through the stanza's interfaces:

```
>>> msg = Message()
>>> msg.values
{'body': '', 'from': None, 'mucnick': None, 'mucroom': None,
 'to': None, 'type': 'normal', 'id': None, 'subject': None}
```

Likewise, assigning to `values` will change the XML content:

```
>>> msg = Message()
>>> msg.values = {'body': 'Hi!', 'to': 'user@example.com'}
>>> msg
'<message to="user@example.com"><body>Hi!</body></message>'
```

New in version 1.0-Beta1.

`init_plugin(attrib, lang=None, existing_xml=None, reuse=True)`

Enable and initialize a stanza plugin.

Parameters `attrib` (*string*) – The `plugin_attrib` value of the plugin to enable.

`interfaces = {'from', 'id', 'payload', 'to', 'type'}`

The set of keys that the stanza provides for accessing and manipulating the underlying XML object. This set may be augmented with the `plugin_attrib` value of any registered stanza plugins.

`is_extension = False`

If you need to add a new interface to an existing stanza, you can create a plugin and set `is_extension` = `True`. Be sure to set the `plugin_attrib` value to the desired interface name, and that it is the only interface listed in `interfaces`. Requests for the new interface from the parent stanza will be passed to the plugin directly.

New in version 1.0-Beta5.

`iterables`

A list of child stanzas whose class is included in `plugin_iterables`.

`keys()`

Return the names of all stanza interfaces provided by the stanza object.

Allows stanza objects to be used like dictionaries.

`lang_interfaces = {}`

New in version 1.1.2.

`match(xpath)`

Compare a stanza object with an XPath-like expression.

If the XPath matches the contents of the stanza object, the match is successful.

The XPath expression may include checks for stanza attributes. For example:

```
'presence@show=xa@priority=2/status'
```

Would match a presence stanza whose show value is set to 'xa', has a priority value of '2', and has a status element.

Parameters `xpath` (*string*) – The XPath expression to check against. It may be either a string or a list of element names with attribute checks.

`name = 'stanza'`

The XML tag name of the element, not including any namespace prefixes. For example, an `ElementBase` object for `<message />` would use `name = 'message'`.

namespace = 'jabber:client'

The XML namespace for the element. Given <foo xmlns="bar" />, then namespace = "bar" should be used. The default namespace is jabber:client since this is being used in an XMPP library.

next()

Return the next iterable stanza.

overrides = []

In some cases you may wish to override the behaviour of one of the parent stanza's interfaces. The overrides list specifies the interface name and access method to be overridden. For example, to override setting the parent's 'condition' interface you would use:

```
overrides = ['set_condition']
```

Getting and deleting the 'condition' interface would not be affected.

New in version 1.0-Beta5.

parent

A weakref.weakref to the parent stanza, if there is one. If not, then *parent* is None.

plugin_attrib = 'plugin'

For *ElementBase* subclasses which are intended to be used as plugins, the plugin_attrib value defines the plugin name. Plugins may be accessed by using the plugin_attrib value as the interface. An example using plugin_attrib = 'foo':

```
register_stanza_plugin(Message, FooPlugin)
msg = Message()
msg['foo']['an_interface_from_the_foo_plugin']
```

plugin_attrib_map = {}

A mapping of the *plugin_attrib* values of registered plugins to their respective classes.

plugin_iterables = {}

The set of stanza classes that can be iterated over using the 'substanzas' interface. Classes are added to this set when registering a plugin with iterable=True:

```
register_stanza_plugin(DiscoInfo, DiscoItem, iterable=True)
```

New in version 1.0-Beta5.

plugin_multi_attrib = ''

For *ElementBase* subclasses that are intended to be an iterable group of items, the plugin_multi_attrib value defines an interface for the parent stanza which returns the entire group of matching substanzas. So the following are equivalent:

```
# Given stanza class Foo, with plugin_multi_attrib = 'foos'
parent['foos']
filter(isinstance(item, Foo), parent['substanzas'])
```

plugin_overrides = {}

A map of interface operations to the overriding functions. For example, after overriding the set operation for the interface body, *plugin_overrides* would be:

```
{'set_body': <some function>}
```

plugin_tag_map = {}

A mapping of root element tag names (in '{namespace}elementname' format) to the plugin classes responsible for them.

plugins

An ordered dictionary of plugin stanzas, mapped by their `plugin_attrib` value.

pop (index=0)

Remove and return the last stanza in the list of iterable substanzas.

Allows stanza objects to be used like lists.

Parameters `index` (`int`) – The index of the stanza to remove.

set_stanza_values (values)

Set multiple stanza interface values using a dictionary.

Stanza plugin values may be set using nested dictionaries.

Parameters `values` – A dictionary mapping stanza interface with values. Plugin interfaces may accept a nested dictionary that will be used recursively.

New in version 1.0-Beta1.

setup (xml=None)

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

sub_interfaces = {}

A subset of `interfaces` which maps interfaces to direct subelements of the underlying XML object. Using this set, the text of these subelements may be set, retrieved, or removed without needing to define custom methods.

tag

The name of the tag for the stanza's root element. It is the same as calling `tag_name()` and is formatted as '`{namespace}elementname`'.

classmethod tag_name()

Return the namespaced name of the stanza's root element.

The format for the tag name is:

```
'{namespace}elementname'
```

For example, for the stanza `<foo xmlns="bar" />`, `stanza.tag_name()` would return "`{bar}foo`".

property values

Return A JSON/dictionary version of the XML content exposed through the stanza's interfaces:

```
>>> msg = Message()
>>> msg.values
{'body': '', 'from': '', 'mucnick': '', 'mucroom': '',
'to': '', 'type': 'normal', 'id': '', 'subject': ''}
```

Likewise, assigning to `values` will change the XML content:

```
>>> msg = Message()
>>> msg.values = {'body': 'Hi!', 'to': 'user@example.com'}
>>> msg
'<message to="user@example.com"><body>Hi!</body></message>'
```

New in version 1.0-Beta1.

`xml`

The underlying XML object for the stanza. It is a standard `xml.etree.ElementTree` object.

`xml_ns = 'http://www.w3.org/XML/1998/namespace'`

The default XML namespace: `http://www.w3.org/XML/1998/namespace`.

6.7.4 StanzaBase

```
class slixmpp.xmlstream.stanzabase.StanzaBase(stream=None, xml=None, stype=None,
                                                sto=None, sfrom=None, sid=None, parent=None)
```

`StanzaBase` provides the foundation for all other stanza objects used by Slixmpp, and defines a basic set of interfaces common to nearly all stanzas. These interfaces are the '`id`', '`type`', '`to`', and '`from`' attributes. An additional interface, '`payload`', is available to access the XML contents of the stanza. Most stanza objects will provide more specific interfaces, however.

Stanza Interfaces:

- id** An optional id value that can be used to associate stanzas
- to** A JID object representing the recipient's JID.
- from** A JID object representing the sender's JID. with their replies.
- type** The type of stanza, typically will be '`normal`', '`error`', '`get`', or '`set`', etc.
- payload** The XML contents of the stanza.

Parameters

- **stream** (`XMLStream`) – Optional `slixmpp.xmlstream.XMLStream` object responsible for sending this stanza.
- **xml** (`XML`) – Optional XML contents to initialize stanza values.
- **stype** (`string`) – Optional stanza type value.
- **sto** – Optional string or `slixmpp.xmlstream.JID` object of the recipient's JID.
- **sfrom** – Optional string or `slixmpp.xmlstream.JID` object of the sender's JID.
- **sid** (`string`) – Optional ID value for the stanza.
- **parent** – Optionally specify a parent stanza object will contain this stanza.

`del_payload()`

Remove the XML contents of the stanza.

`error()`

Set the stanza's type to '`error`'.

`exception(e)`

Handle exceptions raised during stanza processing.

Meant to be overridden.

`get_from()`

Return the value of the stanza's '`from`' attribute.

`get_payload()`

Return a list of XML objects contained in the stanza.

```
get_to()
    Return the value of the stanza's 'to' attribute.

namespace = 'jabber:client'
    The default XMPP client namespace

reply(clear=True)
    Prepare the stanza for sending a reply.

    Swaps the 'from' and 'to' attributes.

    If clear=True, then also remove the stanza's contents to make room for the reply content.

    For client streams, the 'from' attribute is removed.

    Parameters clear (bool) – Indicates if the stanza's contents should be removed. Defaults to True.

send()
    Queue the stanza to be sent on the XML stream.

set_from(value)
    Set the 'from' attribute of the stanza.

    Parameters from (str or JID) – A string or JID object representing the sender's JID.

set_payload(value)
    Add XML content to the stanza.

    Parameters value – Either an XML or a stanza object, or a list of XML or stanza objects.

set_to(value)
    Set the 'to' attribute of the stanza.

    Parameters value – A string or slixmpp.xmlstream.JID object representing the recipient's JID.

set_type(value)
    Set the stanza's 'type' attribute.

    Only type values contained in types are accepted.

    Parameters value (str) – One of the values contained in types

unhandled()
    Called if no handlers have been registered to process this stanza.

    Meant to be overridden.
```

6.8 Stanza Handlers

6.8.1 The Basic Handler

```
class slixmpp.xmlstream.handler.base.BaseHandler(name, matcher, stream=None)
    Base class for stream handlers. Stream handlers are matched with incoming stanzas so that the stanza may be processed in some way. Stanzas may be matched with multiple handlers.

    Handler execution may take place in two phases: during the incoming stream processing, and in the main event loop. The prerun() method is executed in the first case, and run() is called during the second.
```

Parameters

- **name** (*string*) – The name of the handler.
- **matcher** – A *MatcherBase* derived object that will be used to determine if a stanza should be accepted by this handler.
- **stream** – The *XMLStream* instance that the handle will respond to.

check_delete ()

Check if the handler should be removed from the list of stream handlers.

match (xml)

Compare a stanza or XML object with the handler's matcher.

Parameters **xml** – An XML or *ElementBase* object

name

The name of the handler

prerun (payload)

Prepare the handler for execution while the XML stream is being processed.

Parameters **payload** – A *ElementBase* object.

run (payload)

Execute the handler after XML stream processing and during the main event loop.

Parameters **payload** – A *ElementBase* object.

stream

The XML stream this handler is assigned to

6.8.2 Callback

```
class slixmpp.xmlstream.handler.Callback(name, matcher, pointer, thread=False,  
                                         once=False, instream=False, stream=None)
```

The Callback handler will execute a callback function with matched stanzas.

The handler may execute the callback either during stream processing or during the main event loop.

Callback functions are all executed in the same thread, so be aware if you are executing functions that will block for extended periods of time. Typically, you should signal your own events using the Slixmpp object's *event ()* method to pass the stanza off to a threaded event handler for further processing.

Parameters

- **name** (*string*) – The name of the handler.
- **matcher** – A *MatcherBase* derived object for matching stanza objects.
- **pointer** – The function to execute during callback.
- **thread** (*bool*) – **DEPRECATED.** Remains only for backwards compatibility.
- **once** (*bool*) – Indicates if the handler should be used only once. Defaults to False.
- **instream** (*bool*) – Indicates if the callback should be executed during stream processing instead of in the main event loop.
- **stream** – The *XMLStream* instance this handler should monitor.

prerun (payload)

Execute the callback during stream processing, if the callback was created with *instream=True*.

Parameters **payload** – The matched *ElementBase* object.

run (*payload, instream=False*)

Execute the callback function with the matched stanza payload.

Parameters

- **payload** – The matched *ElementBase* object.
- **instream** (*bool*) – Force the handler to execute during stream processing. This should only be used by *prerun()*. Defaults to False.

6.8.3 CoroutineCallback

```
class slixmpp.xmlstream.handler.CoroutineCallback(name, matcher, pointer,
                                                    once=False, instream=False,
                                                    stream=None)
```

The Callback handler will execute a callback function with matched stanzas.

The handler may execute the callback either during stream processing or during the main event loop.

The event will be scheduled to be run soon in the event loop instead of immediately.

Parameters

- **name** (*string*) – The name of the handler.
- **matcher** – A *MatcherBase* derived object for matching stanza objects.
- **pointer** – The function to execute during callback. If **pointer** is not a coroutine, this function will raise a *ValueError*.
- **once** (*bool*) – Indicates if the handler should be used only once. Defaults to False.
- **instream** (*bool*) – Indicates if the callback should be executed during stream processing instead of in the main event loop.
- **stream** – The *XMLStream* instance this handler should monitor.

prerun (*payload*)

Execute the callback during stream processing, if the callback was created with *instream=True*.

Parameters **payload** – The matched *ElementBase* object.

run (*payload, instream=False*)

Execute the callback function with the matched stanza payload.

Parameters

- **payload** – The matched *ElementBase* object.
- **instream** (*bool*) – Force the handler to execute during stream processing. This should only be used by *prerun()*. Defaults to False.

6.8.4 Waiter

```
class slixmpp.xmlstream.handler.Waiter(name, matcher, stream=None)
```

The Waiter handler allows an event handler to block until a particular stanza has been received. The handler will either be given the matched stanza, or `False` if the waiter has timed out.

Parameters

- **name** (*string*) – The name of the handler.
- **matcher** – A `MatcherBase` derived object for matching stanza objects.
- **stream** – The `XMLStream` instance this handler should monitor.

`check_delete()`

Always remove waiters after use.

`prerun(payload)`

Store the matched stanza when received during processing.

Parameters `payload` – The matched `ElementBase` object.

`run(payload)`

Do not process this handler during the main event loop.

`async wait(timeout=None)`

Block an event handler while waiting for a stanza to arrive.

Be aware that this will impact performance if called from a non-threaded event handler.

Will return either the received stanza, or `False` if the waiter timed out.

Parameters `timeout` (*int*) – The number of seconds to wait for the stanza to arrive. Defaults to the the stream's `response_timeout` value.

6.9 Stanza Matchers

6.9.1 The Basic Matcher

```
class slixmpp.xmlstream.matcher.base.MatcherBase(criteria)
```

Base class for stanza matchers. Stanza matchers are used to pick stanzas out of the XML stream and pass them to the appropriate stream handlers.

Parameters `criteria` – Object to compare some aspect of a stanza against.

`match(xml)`

Check if a stanza matches the stored criteria.

Meant to be overridden.

6.9.2 ID Matching

```
class slixmpp.xmlstream.matcher.id.MatcherId(criteria)
```

The ID matcher selects stanzas that have the same stanza ‘id’ interface value as the desired ID.

```
match(xml)
```

Compare the given stanza’s ‘id’ attribute to the stored id value.

Parameters `xml` – The `ElementBase` stanza to compare against.

6.9.3 Stanza Path Matching

```
class slixmpp.xmlstream.matcher.stanzapath.StanzaPath(criteria)
```

The StanzaPath matcher selects stanzas that match a given “stanza path”, which is similar to a normal XPath except that it uses the interfaces and plugins of the stanza instead of the actual, underlying XML.

Parameters `criteria` – Object to compare some aspect of a stanza against.

```
match(stanza)
```

Compare a stanza against a “stanza path”. A stanza path is similar to an XPath expression, but uses the stanza’s interfaces and plugins instead of the underlying XML. See the documentation for the stanza `match()` method for more information.

Parameters `stanza` – The `ElementBase` stanza to compare against.

6.9.4 XPath

```
class slixmpp.xmlstream.matcher.xpath.MatchXPath(criteria)
```

The XPath matcher selects stanzas whose XML contents matches a given XPath expression.

Warning: Using this matcher may not produce expected behavior when using attribute selectors. For Python 2.6 and 3.1, the `ElementTree find()` method does not support the use of attribute selectors. If you need to support Python 2.6 or 3.1, it might be more useful to use a `StanzaPath` matcher.

If the value of `IGNORE_NS` is set to `True`, then XPath expressions will be matched without using namespaces.

```
match(xml)
```

Compare a stanza’s XML contents to an XPath expression.

If the value of `IGNORE_NS` is set to `True`, then XPath expressions will be matched without using namespaces.

Warning: In Python 2.6 and 3.1 the `ElementTree find()` method does not support attribute selectors in the XPath expression.

Parameters `xml` – The `ElementBase` stanza to compare against.

6.9.5 XMLMask

```
class slixmpp.xmlstream.matcher.xmlmask.MatchXMLMask (criteria, de-  

fault_ns='jabber:client')
```

The XMLMask matcher selects stanzas whose XML matches a given XML pattern, or mask. For example, message stanzas with body elements could be matched using the mask:

```
<message xmlns="jabber:client"><body /></message>
```

Use of XMLMask is discouraged, and *MatchXPath* or *StanzaPath* should be used instead.

Parameters criteria – Either an *Element* XML object or XML string to use as a mask.

match (*xml*)

Compare a stanza object or XML object against the stored XML mask.

Overrides MatcherBase.match.

Parameters xml – The stanza object or XML object to compare against.

setDefaultNS (*ns*)

Set the default namespace to use during comparisons.

Parameters ns – The new namespace to use as the default.

6.10 XML Stream

```
class slixmpp.xmlstream.xmlstream.XMLStream (host=', port=0)
```

An XML stream connection manager and event dispatcher.

The XMLStream class abstracts away the issues of establishing a connection with a server and sending and receiving XML “stanzas”. A stanza is a complete XML element that is a direct child of a root document element. Two streams are used, one for each communication direction, over the same socket. Once the connection is closed, both streams should be complete and valid XML documents.

Three types of events are provided to manage the stream:

Stream Triggered based on received stanzas, similar in concept to events in a SAX XML parser.

Custom Triggered manually.

Scheduled Triggered based on time delays.

Typically, stanzas are first processed by a stream event handler which will then trigger custom events to continue further processing, especially since custom event handlers may run in individual threads.

Parameters

- **socket** – Use an existing socket for the stream. Defaults to None to generate a new socket.
- **host** (*string*) – The name of the target server.
- **port** (*int*) – The port to use for the connection. Defaults to 0.

abort()

Forcibly close the connection

add_event_handler (*name*, *pointer*, *disposable=False*)

Add a custom event handler that will be executed whenever its event is manually triggered.

Parameters

- **name** – The name of the event that will trigger this handler.

- **pointer** – The function to execute.
- **disposable** – If set to True, the handler will be discarded after one use. Defaults to False.

`add_filter(mode, handler, order=None)`

Add a filter for incoming or outgoing stanzas.

These filters are applied before incoming stanzas are passed to any handlers, and before outgoing stanzas are put in the send queue.

Each filter must accept a single stanza, and return either a stanza or None. If the filter returns None, then the stanza will be dropped from being processed for events or from being sent.

Parameters

- **mode** – One of 'in' or 'out'.
- **handler** – The filter function.
- **order (int)** – The position to insert the filter in the list of active filters.

`address`

The desired, or actual, address of the connected server.

`ca_certs`

Path to a file containing certificates for verifying the server SSL certificate. A non-None value will trigger certificate checking.

Note: On Mac OS X, certificates in the system keyring will be consulted, even if they are not in the provided file.

`cancel_connection_attempt()`

Immediately cancel the current `create_connection()` Future. This is useful when a client using slixmpp tries to connect on flaky networks, where sometimes a connection just gets lost and it needs to reconnect while the attempt is still ongoing.

`certfile`

Path to a file containing a client certificate to use for authenticating via SASL EXTERNAL. If set, there must also be a corresponding `:attr:keyfile` value.

`ciphers`

The list of accepted ciphers, in OpenSSL Format. It might be useful to override it for improved security over the python defaults.

`configure_dns(resolver, domain=None, port=None)`

Configure and set options for a Resolver instance, and other DNS related tasks. For example, you can also check `getaddrinfo()` to see if you need to call out to `libresolv.so.2` to run `res_init()`.

Meant to be overridden.

Parameters

- **resolver** – A Resolver instance or None if dnspython is not installed.
- **domain** – The initial domain under consideration.
- **port** – The initial port under consideration.

`configure_socket()`

Set timeout and other options for self.socket.

Meant to be overridden.

connect (*host*='', *port*=0, *use_ssl*=False, *force_starttls*=True, *disable_starttls*=False)

Create a new socket and connect to the server.

Parameters

- **host** – The name of the desired server for the connection.
- **port** – Port to connect to on the server.
- **use_ssl** – Flag indicating if SSL should be used by connecting directly to a port using SSL. If it is False, the connection will be upgraded to SSL/TLS later, using STARTTLS. Only use this value for old servers that have specific port for SSL/TLS
- **force_starttls** – If True, the connection will be aborted if the server does not initiate a STARTTLS negotiation. If None, the connection will be upgraded to TLS only if the server initiate the STARTTLS negotiation, otherwise it will connect in clear. If False it will never upgrade to TLS, even if the server provides it. Use this for example if you're on localhost

connection_lost (*exception*)

On any kind of disconnection, initiated by us or not. This signals the closure of the TCP connection

connection_made (*transport*)

Called when the TCP connection has been established with the server

data_received (*data*)

Called when incoming data is received on the socket.

We feed that data to the parser and the see if this produced any XML event. This could trigger one or more event (a stanza is received, the stream is opened, etc).

default_domain

The domain to try when querying DNS records.

default_ns

The default namespace of the stream content, not of the stream wrapper itself.

default_port

The default port to return when querying DNS records.

del_event_handler (*name*, *pointer*)

Remove a function as a handler for an event.

Parameters

- **name** – The name of the event.
- **pointer** – The function to remove as a handler.

del_filter (*mode*, *handler*)

Remove an incoming or outgoing filter.

disconnect (*wait*=2.0, *reason*=None, *ignore_send_queue*=False)

Close the XML stream and wait for an acknowledgement from the server for at most *wait* seconds. After the given number of seconds has passed without a response from the server, or when the server successfully responds with a closure of its own stream, *abort()* is called. If *wait* is 0.0, this will call *abort()* directly without closing the stream.

Does nothing but trigger the disconnected event if we are not connected.

Parameters

- **wait** (Union[float, int]) – Time to wait for a response from the server.
- **reason** (Optional[str]) – An optional reason for the disconnect.

- **ignore_send_queue** (bool) – Boolean to toggle if we want to ignore the in-flight stanzas and disconnect immediately.

Return type Future

Returns A future that ends when all code involved in the disconnect has ended

disconnect_reason

The reason why we are disconnecting from the server

disconnected: _asyncio.Future

An asyncio Future being done when the stream is disconnected.

dns_answers

A list of DNS results that have not yet been tried.

dns_service

The service name to check with DNS SRV records. For example, setting this to 'xmpp-client' would query the _xmpp-client._tcp service.

end_session_on_disconnect

Flag for controlling if the session can be considered ended if the connection is terminated.

eof_received()

When the TCP connection is properly closed by the remote end

event (name, data={})

Manually trigger a custom event.

Parameters

- **name** – The name of the event to trigger.
- **data** – Data that will be passed to each event handler. Defaults to an empty dictionary, but is usually a stanza object.

event_handled(name)

Returns the number of registered handlers for an event.

Parameters name – The name of the event to check.

exception(exception)

Process an unknown exception.

Meant to be overridden.

Parameters exception – An unhandled exception object.

async get_dns_records(domain, port=None)

Get the DNS records for a domain.

Parameters

- **domain** – The domain in question.
- **port** – If the results don't include a port, use this one.

get_ssl_context()

Get SSL context.

incoming_filter(xml)

Filter incoming XML objects before they are processed.

Possible uses include remapping namespaces, or correcting elements from sources with incorrect behavior.

Meant to be overridden.

init_parser()
init the XML parser. The parser must always be reset for each new connexion

keyfile
Path to a file containing the private key for the selected client certificate to use for authenticating via SASL EXTERNAL.

namespace_map
A mapping of XML namespaces to well-known prefixes.

new_id()
Generate and return a new stream ID in hexadecimal form.
Many stanzas, handlers, or matchers may require unique ID values. Using this method ensures that all new ID values are unique in this stream.

async pick_dns_answer(domain, port=None)
Pick a server and port from DNS answers.
Gets DNS answers if none available. Removes used answer from available answers.

Parameters

- **domain** – The domain in question.
- **port** – If the results don't include a port, use this one.

process(*, forever=True, timeout=None)
Process all the available XMPP events (receiving or sending data on the socket(s), calling various registered callbacks, calling expired timers, handling signal events, etc). If timeout is None, this function will run forever. If timeout is a number, this function will return after the given time in seconds.

proxy_config
An optional dictionary of proxy settings. It may provide: :host: The host offering proxy services. :port: The port for the proxy service. :username: Optional username for accessing the proxy. :password: Optional password for accessing the proxy.

reconnect(wait=2.0, reason='Reconnecting')
Calls disconnect(), and once we are disconnected (after the timeout, or when the server acknowledgement is received), call connect()

register_handler(handler, before=None, after=None)
Add a stream event handler that will be executed when a matching stanza is received.
Parameters **handler** – The *BaseHandler* derived object to execute.

register_stanza(stanza_class)
Add a stanza object class as a known root stanza.
A root stanza is one that appears as a direct child of the stream's root element.
Stanzas that appear as substanzas of a root stanza do not need to be registered here. That is done using register_stanza_plugin() from slixmpp.xmlstream.stanzabase.
Stanzas that are not registered will not be converted into stanza objects, but may still be processed using handlers and matchers.
Parameters **stanza_class** – The top-level stanza object's class.

remove_handler(name)
Remove any stream event handlers with the given name.
Parameters **name** – The name of the handler.

remove_stanza (*stanza_class*)

Remove a stanza from being a known root stanza.

A root stanza is one that appears as a direct child of the stream's root element.

Stanzas that are not registered will not be converted into stanza objects, but may still be processed using handlers and matchers.

async run_filters()

Background loop that processes stanzas to send.

schedule (*name*, *seconds*, *callback*, *args=()*, *kwargs={}*, *repeat=False*)

Schedule a callback function to execute after a given delay.

Parameters

- **name** – A unique name for the scheduled callback.
- **seconds** – The time in seconds to wait before executing.
- **callback** – A pointer to the function to execute.
- **args** – A tuple of arguments to pass to the function.
- **kwargs** – A dictionary of keyword arguments to pass to the function.
- **repeat** – Flag indicating if the scheduled event should be reset and repeat after executing.

send (*data*, *use_filters=True*)

A wrapper for [send_raw\(\)](#) for sending stanza objects.

Parameters

- **data** – The [ElementBase](#) stanza to send on the stream.
- **use_filters** (*bool*) – Indicates if outgoing filters should be applied to the given stanza data. Disabling filters is useful when resending stanzas. Defaults to True.

send_raw (*data*)

Send raw data across the stream.

Parameters **data** (*string*) – Any bytes or utf-8 string value.

send_xml (*data*)

Send an XML object on the stream

Parameters **data** – The [Element](#) XML object to send on the stream.

start_stream_handler (*xml*)

Perform any initialization actions, such as handshakes, once the stream header has been sent.

Meant to be overridden.

async start_tls()

Perform handshakes for TLS.

If the handshake is successful, the XML stream will need to be restarted.

stream_footer

The default closing tag for the stream element.

stream_header

The default opening tag for the stream element.

stream_ns

The namespace of the enveloping stream element.

use_aiodns

If set to `True`, allow using the `dnspython` DNS library if available. If set to `False`, the builtin DNS resolver will be used, even if `dnspython` is installed.

use_cdata

Use CDATA for escaping instead of XML entities. Defaults to `False`.

use_ipv6

If set to `True`, attempt to use IPv6.

use_proxy

If set to `True`, attempt to connect through an HTTP proxy based on the settings in `proxy_config`.

use_ssl

Enable connecting to the server directly over SSL, in particular when the service provides two ports: one for non-SSL traffic and another for SSL traffic.

async wait_until (event, timeout=30)

Utility method to wake on the next firing of an event. (Registers a disposable handler on it)

Parameters

- **event** (`str`) – Event to wait on.
- **timeout** (`int`) – Timeout

Return type Any**whitespace_keepalive**

If `True`, periodically send a whitespace character over the wire to keep the connection alive. Mainly useful for connections traversing NAT.

whitespace_keepalive_interval

The default interval between keepalive signals when `whitespace_keepalive` is enabled.

6.11 XML Serialization

Since the XML layer of Slixmpp is based on `ElementTree`, why not just use the built-in `tostring()` method? The answer is that using that method produces ugly results when using namespaces. The `tostring()` method used here intelligently hides namespaces when able and does not introduce excessive namespace prefixes:

```
>>> from slixmpp.xmlstream.tostring import tostring
>>> from xml.etree import ElementTree as ET
>>> xml = ET.fromstring('<foo xmlns="bar"><baz /></foo>')
>>> ET.tostring(xml)
'<ns0:foo xmlns:ns0="bar"><ns0:baz /></ns0:foo>'
>>> tostring(xml)
'<foo xmlns="bar"><baz /></foo>'
```

As a side effect of this namespace hiding, using `tostring()` may produce unexpected results depending on how the `tostring()` method is invoked. For example, when sending XML on the wire, the main XMPP stanzas with their namespace of `jabber:client` will not include the namespace because that is already declared by the stream header. But, if you create a `Message` instance and dump it to the terminal, the `jabber:client` namespace will appear.

```
slixmpp.xmlstream.tostring(xml=None, xmlns='', stream=None, outbuffer='', top_level=False, open_only=False, namespaces=None)
```

Serialize an XML object to a Unicode string.

If an outer xmlns is provided using `xmlns`, then the current element's namespace will not be included if it matches the outer namespace. An exception is made for elements that have an attached stream, and appear at the stream root.

Parameters

- `xml` (`Element`) – The XML object to serialize.
- `xmlns` (`string`) – Optional namespace of an element wrapping the XML object.
- `stream` (`XMLStream`) – The XML stream that generated the XML object.
- `outbuffer` (`string`) – Optional buffer for storing serializations during recursive calls.
- `top_level` (`bool`) – Indicates that the element is the outermost element.
- `namespaces` (`set`) – Track which namespaces are in active use so that new ones can be declared when needed.

Return type Unicode string

6.11.1 Escaping Special Characters

In order to prevent errors when sending arbitrary text as the textual content of an XML element, certain characters must be escaped. These are: &, <, >, ", and '. The default escaping mechanism is to replace those characters with their equivalent escape entities: &, <, >, ', and ".

In the future, the use of CDATA sections may be allowed to reduce the size of escaped text or for when other XMPP processing agents do not understand these entities.

6.12 Plugins

6.12.1 Plugin index

XEP 0004

```
class slixmpp.plugins.xep_0004.XEP_0004(xmpp, config=None)
XEP-0004: Data Forms
stanza = <module 'slixmpp.plugins.xep_0004.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0004.stanza.field.FieldOption(xml=None, parent=None)

interfaces = {'label', 'value'}
name = 'option'
namespace = 'jabber:x:data'
plugin_attrib = 'option'
```

```
plugin_multi_attrib = 'options'
sub_interfaces = {'value'}

class slixmpp.plugins.xep_0004.stanza.field.FormField(xml=None, parent=None)

    addOption(label='', value='')
    add_option(label='', value='')
    delOptions()
    delRequired()
    delValue()
    del_options()
    del_required()
    del_value()

    field_types = {'boolean', 'fixed', 'hidden', 'jid-multi', 'jid-single', 'list-multi',
                   'list-single'}
    getAnswer()
    getOptions()
    getRequired()
    getValue(convert=True)
    get_answer()
    get_options()
    get_required()
    get_value(convert=True)

    interfaces = {'answer', 'desc', 'label', 'required', 'type', 'value', 'var'}
    multi_line_types = {'hidden', 'text-multi'}
    multi_value_types = {'hidden', 'jid-multi', 'list-multi', 'text-multi'}
    name = 'field'
    namespace = 'jabber:x:data'
    option_types = {'list-multi', 'list-single'}
    plugin_attrib = 'field'
    plugin_attrib_map = {}
    plugin_multi_attrib = 'fields'
    plugin_tag_map = {}

    setAnswer(answer)
    setFalse()
    setOptions(options)
    setRequired(required)
    setTrue()
```

```
setValue (value)
set_answer (answer)
set_false ()
set_options (options)
set_required (required)
set_true ()
set_type (value)
set_value (value)
setup (xml=None)
    Initialize the stanza's XML contents.

    Will return True if XML was generated according to the stanza's definition instead of building a stanza
    object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
    new XML.

sub_interfaces = {'desc'}
true_values = {True, '1', 'true'}
```

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0004.stanza.form.Form(*args, **kwargs)

    addField(var='', ftype=None, label='', desc='', required=False, value=None, options=None,
             **kwargs)
    addReported(var,ftype=None,label='',desc='',**kwargs)
    add_field(var='', ftype=None, label='', desc='', required=False, value=None, options=None,
              **kwargs)
    add_item(values)
    add_reported(var,ftype=None,label='',desc='',**kwargs)
    cancel()
    delFields()
    delInstructions()
    delReported()
    del_fields()
    del_instructions()
    del_items()
    del_reported()
    property field
    form_types = {'cancel', 'form', 'result', 'submit'}
    getFields(use_dict=False)
```

```

getInstructions()
getReported()
getValues()
get_fields(use_dict=False)
get_instructions()
get_items()
get_reported()
get_values()
interfaces = OrderedSet(['instructions', 'reported', 'title', 'type', 'items', 'values'])
merge(other)
name = 'x'
namespace = 'jabber:x:data'
plugin_attrib = 'form'
reply()
setFields(fields)
setInstructions(instructions)
setReported(report)
    This either needs a dictionary of dictionaries or a dictionary of form fields. :param report: :return:
setValues(values)
set_fields(fields)
set_instructions(instructions)
set_items(items)
set_reported(report)
    This either needs a dictionary of dictionaries or a dictionary of form fields. :param report: :return:
set_type(ftype)
set_values(values)
setup(xml=None)
    Initialize the stanza's XML contents.
    Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.
    Parameters xml – An existing XML object to use for the stanza's content instead of generating new XML.
sub_interfaces = {'title'}

```

XEP 0009

```
class slixmpp.plugins.xep_0009.XEP_0009(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0009.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Dann Martens (TOMOTON). This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0009.stanza.RPC.MethodCall(xml=None, parent=None)
```

```
get_method_name()
get_params()
interfaces = {'method_name', 'params'}
name = 'methodCall'
namespace = 'jabber:iq:rpc'
plugin_attrib = 'method_call'
plugin_attrib_map = {}
plugin_tag_map = {}
set_method_name(value)
set_params(params)
subinterfaces = {}
```

```
class slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse(xml=None, parent=None)
```

```
get_fault()
get_params()
interfaces = {'fault', 'params'}
name = 'methodResponse'
namespace = 'jabber:iq:rpc'
plugin_attrib = 'method_response'
plugin_attrib_map = {}
plugin_tag_map = {}
set_fault(fault)
set_params(params)
subinterfaces = {}
```

```
class slixmpp.plugins.xep_0009.stanza.RPCQuery(xml=None, parent=None)
```

```
interfaces = {}
name = 'query'
namespace = 'jabber:iq:rpc'
plugin_attrib = 'rpc_query'
plugin_attrib_map = {}
plugin_tag_map = {}
subinterfaces = {}
```

XEP 0012

```
class slixmpp.plugins.xep_0012.XEP_0012(xmpp, config=None)
```

XEP-0012 Last Activity

```
stanza = <module 'slixmpp.plugins.xep_0012.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0012.stanza.LastActivity(xml=None, parent=None)
```

```
del_status()
get_seconds()
get_status()
interfaces = {'seconds', 'status'}
name = 'query'
namespace = 'jabber:iq:last'
plugin_attrib = 'last_activity'
set_seconds(value)
set_status(value)
```

XEP 0013

```
class slixmpp.plugins.xep_0013.XEP_0013(xmpp, config=None)
```

XEP-0013 Flexible Offline Message Retrieval

```
stanza = <module 'slixmpp.plugins.xep_0013.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0013.stanza.Item(xml=None, parent=None)

    actions = {'remove', 'view'}

    get_jid()

    interfaces = {'action', 'jid', 'node'}
    name = 'item'
    namespace = 'http://jabber.org/protocol/offline'
    plugin_attrib = 'item'
    set_jid(value)

class slixmpp.plugins.xep_0013.stanza.Offline(xml=None, parent=None)

    bool_interfaces = {'fetch', 'purge', 'results'}

    del_results()
    get_results()

    interfaces = {'fetch', 'purge', 'results'}
    name = 'offline'
    namespace = 'http://jabber.org/protocol/offline'
    plugin_attrib = 'offline'
    plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0013.stanza.Item'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0013.stanza.Item'>}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/offline}item': <class 'slixmpp.plugins.xep_0013.stanza.Item'>}
    set_results(values)

    setup(xml=None)
        Initialize the stanza's XML contents.

        Will return True if XML was generated according to the stanza's definition instead of building a stanza
        object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
        new XML.
```

XEP 0020

```
class slixmpp.plugins.xep_0020.XEP_0020(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0020.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2013 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0020.stanza.FeatureNegotiation(xml=None, parent=None)

interfaces = {}
name = 'feature'
namespace = 'http://jabber.org/protocol/feature-neg'
plugin_attrib = 'feature_neg'
```

XEP 0027

```
class slixmpp.plugins.xep_0027.XEP_0027(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0027.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0027.stanza.Encrypted(xml=None, parent=None)
```

```
get_encrypted()
interfaces = {'encrypted'}
is_extension = True
name = 'x'
namespace = 'jabber:x:encrypted'
plugin_attrib = 'encrypted'
set_encrypted(value)
```

```
class slixmpp.plugins.xep_0027.stanza.Signed(xml=None, parent=None)
```

```
get_signed()
interfaces = {'signed'}
is_extension = True
```

```
name = 'x'
namespace = 'jabber:x:signed'
plugin_attrib = 'signed'
set_signed(value)
```

XEP 0030

```
class slixmpp.plugins.xep_0030.XEP_0030(xmpp, config=None)
XEP-0030: Service Discovery
```

Service discovery in XMPP allows entities to discover information about other agents in the network, such as the feature sets supported by a client, or signposts to other, related entities.

Also see <<http://www.xmpp.org/extensions/xep-0030.html>>.

The XEP-0030 plugin works using a hierarchy of dynamic node handlers, ranging from global handlers to specific JID+node handlers. The default set of handlers operate in a static manner, storing disco information in memory. However, custom handlers may use any available backend storage mechanism desired, such as SQLite or Redis.

Node handler hierarchy:

JID		Node		Level
<hr/>				
None		None		Global
Given		None		All nodes for the JID
None		Given		Node on <code>self.xmpp.boundjid</code>
Given		Given		A single node

Stream Handlers:

```
Disco Info -- Any Iq stanza that includes a query with the
              namespace http://jabber.org/protocol/disco#info.
Disco Items -- Any Iq stanza that includes a query with the
              namespace http://jabber.org/protocol/disco#items.
```

Events:

```
disco_info      -- Received a disco#info Iq query result.
disco_items     -- Received a disco#items Iq query result.
disco_info_query -- Received a disco#info Iq query request.
disco_items_query -- Received a disco#items Iq query request.
```

Attributes:

Variables

- **static** – Object containing the default set of static node handlers.
- **default_handlers** – A dictionary mapping operations to the default global handler (by default, the static handlers).

add_feature (*feature*, *node=None*, *jid=None*)

Add a feature to a JID/node combination.

Parameters

- **feature** (*str*) – The namespace of the supported feature.

- **node** (Optional[str]) – The node to modify.
- **jid** (Optional[JID]) – The JID to modify.

add_identity (category='', itype='', name='', node=None, jid=None, lang=None)

Add a new identity to the given JID/node combination.

Each identity must be unique in terms of all four identity components: category, type, name, and language.

Multiple, identical category/type pairs are allowed only if the xml:lang values are different. Likewise, multiple category/type/xml:lang pairs are allowed so long as the names are different. A category and type is always required.

Parameters

- **category** – The identity's category.
- **itype** – The identity's type.
- **name** – Optional name for the identity.
- **lang** – Optional two-letter language code.
- **node** – The node to modify.
- **jid** – The JID to modify.

add_item (jid='', name='', node=None, subnode='', ijid=None)

Add a new item element to the given JID/node combination.

Each item is required to have a JID, but may also specify a node value to reference non-addressable entities.

Parameters

- **jid** – The JID for the item.
- **name** – Optional name for the item.
- **node** – The node to modify.
- **subnode** – Optional node for the item.
- **ijid** – The JID to modify.

del_feature (jid=None, node=None, **kwargs)

Remove a feature from a given JID/node combination.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **feature** – The feature's namespace.

del_features (jid=None, node=None, **kwargs)

Remove all features from a JID/node combination.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.

del_identities (jid=None, node=None, **kwargs)

Remove all identities for a JID/node combination.

If a language is specified, only identities using that language will be removed.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **lang** – Optional. If given, only remove identities using this xml:lang value.

del_identity (*jid=None, node=None, **kwargs*)

Remove an identity from the given JID/node combination.

Parameters

- **jid** (Optional[*JID*]) – The JID to modify.
- **node** (Optional[str]) – The node to modify.
- **category** – The identity's category.
- **itype** – The identity's type value.
- **name** – Optional, human readable name for the identity.
- **lang** – Optional, the identity's xml:lang value.

del_item (*jid=None, node=None, **kwargs*)

Remove a single item from the given JID/node combination.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **ijid** – The item's JID.
- **inode** – The item's node.

del_items (*jid=None, node=None, **kwargs*)

Remove all items from the given JID/node combination.

Arguments: :param jid: The JID to modify. :param node: Optional node to modify.

del_node_handler (*hype, jid, node*)

Remove a handler type for a JID and node combination.

The next handler in the hierarchy will be used if one exists. If removing the global handler, make sure that other handlers exist to process existing nodes.

Node handler hierarchy:

JID	Node	Level
<hr/>		
None	None	Global
Given	None	All nodes for the JID
None	Given	Node on <code>self.xmpp.boundjid</code>
Given	Given	A single node

Parameters

- **hype** – The type of handler to remove.
- **jid** – The JID from which to remove the handler.
- **node** – The node from which to remove the handler.

get_info (*jid=None, node=None, local=None, cached=None, **kwargs*)

Retrieve the disco#info results from a given JID/node combination.

Info may be retrieved from both local resources and remote agents; the local parameter indicates if the information should be gathered by executing the local node handlers, or if a disco#info stanza must be generated and sent.

If requesting items from a local JID/node, then only a DiscoInfo stanza will be returned. Otherwise, an Iq stanza will be returned.

Parameters

- **jid** – Request info from this JID.
- **node** – The particular node to query.
- **local** – If true, then the query is for a JID/node combination handled by this Slixmpp instance and no stanzas need to be sent. Otherwise, a disco stanza must be sent to the remote JID to retrieve the info.
- **cached** – If true, then look for the disco info data from the local cache system. If no results are found, send the query as usual. The self.use_cache setting must be set to true for this option to be useful. If set to false, then the cache will be skipped, even if a result has already been cached. Defaults to false.

async get_info_from_domain (*domain=None, timeout=None, cached=True, callback=None*)

Fetch disco#info of specified domain and one disco#items level below

get_items (*jid=None, node=None, local=False, **kwargs*)

Retrieve the disco#items results from a given JID/node combination.

Items may be retrieved from both local resources and remote agents; the local parameter indicates if the items should be gathered by executing the local node handlers, or if a disco#items stanza must be generated and sent.

If requesting items from a local JID/node, then only a DiscoItems stanza will be returned. Otherwise, an Iq stanza will be returned.

Parameters

- **jid** – Request info from this JID.
- **node** – The particular node to query.
- **local** – If true, then the query is for a JID/node combination handled by this Slixmpp instance and no stanzas need to be sent. Otherwise, a disco stanza must be sent to the remove JID to retrieve the items.
- **iterator** – If True, return a result set iterator using the XEP-0059 plugin, if the plugin is loaded. Otherwise the parameter is ignored.

has_identity (*jid=None, node=None, category=None, itype=None, lang=None, local=False, cached=True, ifrom=None*)

Check if a JID provides a given identity.

Return values: :param True: The identity is provided :param False: The identity is not listed :param None: Nothing could be found due to a timeout

Parameters

- **jid** – Request info from this JID.
- **node** – The particular node to query.
- **category** – The category of the identity to check.

- **itype** – The type of the identity to check.
- **lang** – The language of the identity to check.
- **local** – If true, then the query is for a JID/node combination handled by this Slixmpp instance and no stanzas need to be sent. Otherwise, a disco stanza must be sent to the remove JID to retrieve the info.
- **cached** – If true, then look for the disco info data from the local cache system. If no results are found, send the query as usual. The self.use_cache setting must be set to true for this option to be useful. If set to false, then the cache will be skipped, even if a result has already been cached. Defaults to false.

restore_defaults (*jid=None, node=None, handlers=None*)

Change all or some of a node's handlers to the default handlers. Useful for manually overriding the contents of a node that would otherwise be handled by a JID level or global level dynamic handler.

The default is to use the built-in static handlers, but that may be changed by modifying self.default_handlers.

Parameters

- **jid** – The JID owning the node to modify.
- **node** – The node to change to using static handlers.
- **handlers** – Optional list of handlers to change to the default version. If provided, only these handlers will be changed. Otherwise, all handlers will use the default version.

set_features (*jid=None, node=None, **kwargs*)

Add or replace the set of supported features for a JID/node combination.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **features** – The new set of supported features.

set_identities (*jid=None, node=None, **kwargs*)

Add or replace all identities for the given JID/node combination.

The identities must be in a set where each identity is a tuple of the form: (category, type, lang, name)

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **identities** – A set of identities in tuple form.
- **lang** – Optional, xml:lang value.

set_info (*jid=None, node=None, info=None*)

Set the disco#info data for a JID/node based on an existing disco#info stanza.

set_items (*jid=None, node=None, **kwargs*)

Set or replace all items for the specified JID/node combination.

The given items must be in a list or set where each item is a tuple of the form: (jid, node, name).

Parameters

- **jid** – The JID to modify.

- **node** – Optional node to modify.
- **items** – A series of items in tuple format.

set_node_handler(*hype*, *jid=None*, *node=None*, *handler=None*)

Add a node handler for the given hierarchy level and handler type.

Node handlers are ordered in a hierarchy where the most specific handler is executed. Thus, a fallback, global handler can be used for the majority of cases with a few node specific handler that override the global behavior.

Node handler hierarchy:

JID	Node	Level
None	None	Global
Given	None	All nodes for the JID
None	Given	Node on <code>self.xmpp.boundjid</code>
Given	Given	A single node

Handler types:

```
get_info
get_items
set_identities
set_features
set_items
del_items
del_identities
del_identity
del_feature
del_features
del_item
add_identity
add_feature
add_item
```

Parameters

- **hype** (`str`) – The operation provided by the handler.
- **jid** (`Optional[JID]`) – The JID the handler applies to. May be narrowed further if a node is given.
- **node** (`Optional[str]`) – The particular node the handler is for. If no JID is given, then the `self.xmpp.boundjid.full` is assumed.
- **handler** (`Optional[Callable]`) – The handler function to use.

stanza = <module 'slixmpp.plugins.xep_0030.stanza' from '/home/docs/checkouts/readthed

supports(*jid=None*, *node=None*, *feature=None*, *local=False*, *cached=True*, *ifrom=None*)

Check if a JID supports a given feature.

Return values: :param True: The feature is supported :param False: The feature is not listed as supported :param None: Nothing could be found due to a timeout

Parameters

- **jid** – Request info from this JID.
- **node** – The particular node to query.

- **feature** – The name of the feature to check.
- **local** – If true, then the query is for a JID/node combination handled by this Slixmpp instance and no stanzas need to be sent. Otherwise, a disco stanza must be sent to the remove JID to retrieve the info.
- **cached** – If true, then look for the disco info data from the local cache system. If no results are found, send the query as usual. The self.use_cache setting must be set to true for this option to be useful. If set to false, then the cache will be skipped, even if a result has already been cached. Defaults to false.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

class slixmpp.plugins.xep_0030.stanza.info.DiscoInfo (*xml=None, parent=None*)

XMPP allows for users and agents to find the identities and features supported by other entities in the XMPP network through service discovery, or “disco”. In particular, the “disco#info” query type for <iq> stanzas is used to request the list of identities and features offered by a JID.

An identity is a combination of a category and type, such as the ‘client’ category with a type of ‘pc’ to indicate the agent is a human operated client with a GUI, or a category of ‘gateway’ with a type of ‘aim’ to identify the agent as a gateway for the legacy AIM protocol. See <<http://xmpp.org/registrar/disco-categories.html>> for a full list of accepted category and type combinations.

Features are simply a set of the namespaces that identify the supported features. For example, a client that supports service discovery will include the feature ‘<http://jabber.org/protocol/disco#info>’.

Since clients and components may operate in several roles at once, identity and feature information may be grouped into “nodes”. If one were to write all of the identities and features used by a client, then node names would be like section headings.

Example disco#info stanzas:

```
<iq type="get">
    <query xmlns="http://jabber.org/protocol/disco#info" />
</iq>

<iq type="result">
    <query xmlns="http://jabber.org/protocol/disco#info">
        <identity category="client" type="bot" name="Slixmpp Bot" />
        <feature var="http://jabber.org/protocol/disco#info" />
        <feature var="jabber:x:data" />
        <feature var="urn:xmpp:ping" />
    </query>
</iq>
```

Stanza Interface:

```
node      -- The name of the node to either
           query or return info from.
identities -- A set of 4-tuples, where each tuple contains
             the category, type, xml:lang, and name
             of an identity.
features   -- A set of namespaces for features.
```

add_feature (*feature*)

Add a single, new feature.

Parameters **feature** – The namespace of the supported feature.

add_identity (*category*, *itype*, *name=None*, *lang=None*)

Add a new identity element. Each identity must be unique in terms of all four identity components.

Multiple, identical category/type pairs are allowed only if the xml:lang values are different. Likewise, multiple category/type/xml:lang pairs are allowed so long as the names are different. In any case, a category and type are required.

Parameters

- **category** – The general category to which the agent belongs.
- **itype** – A more specific designation with the category.
- **name** – Optional human readable name for this identity.
- **lang** – Optional standard xml:lang value.

del_feature (*feature*)

Remove a single feature.

Parameters **feature** – The namespace of the removed feature.

del_features ()

Remove all features.

del_identities (*lang=None*)

Remove all identities. If a language was specified, only remove identities using that language.

Parameters **lang** – Optional, standard xml:lang value.

del_identity (*category*, *itype*, *name=None*, *lang=None*)

Remove a given identity.

Parameters

- **category** – The general category to which the agent belonged.
- **itype** – A more specific designation with the category.
- **name** – Optional human readable name for this identity.
- **lang** – Optional, standard xml:lang value.

get_features (*dedupe=True*)

Return the set of all supported features.

get_identities (*lang=None*, *dedupe=True*)

Return a set of all identities in tuple form as so:

(category, type, lang, name)

If a language was specified, only return identities using that language.

Parameters

- **lang** – Optional, standard xml:lang value.
- **dedupe** – If True, de-duplicate identities, otherwise return a list of all identities.

```
interfaces = {'features', 'identities', 'node'}
```

```
lang_interfaces = {'identities'}
```

```
name = 'query'  
namespace = 'http://jabber.org/protocol/disco#info'  
plugin_attrib = 'disco_info'  
set_features(features)
```

Add or replace the set of supported features.

Parameters **features** – The new set of supported features.

```
set_identities(identities, lang=None)
```

Add or replace all identities. The identities must be a in set where each identity is a tuple of the form:
(category, type, lang, name)

If a language is specified, any identities using that language will be removed to be replaced with the given identities.

Note: An identity's language will not be changed regardless of the value of lang.

Parameters

- **identities** – A set of identities in tuple form.
- **lang** – Optional, standard xml:lang value.

```
setup(xml=None)
```

Populate the stanza object using an optional XML object.

Overrides ElementBase.setup

Caches identity and feature information.

Parameters **xml** – Use an existing XML object for the stanza's values.

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0030.stanza.items.DiscoItem(xml=None, parent=None)
```

```
get_name()
```

Return the item's human readable name, or None.

```
get_node()
```

Return the item's node name or None.

```
interfaces = {'jid', 'name', 'node'}
```

```
name = 'item'
```

```
namespace = 'http://jabber.org/protocol/disco#items'
```

```
plugin_attrib = 'item'
```

```
class slixmpp.plugins.xep_0030.stanza.items.DiscoItems(xml=None, parent=None)
```

Example disco#items stanzas:

```

<iq type="get">
    <query xmlns="http://jabber.org/protocol/disco#items" />
</iq>

<iq type="result">
    <query xmlns="http://jabber.org/protocol/disco#items">
        <item jid="chat.example.com"
              node="xmppdev"
              name="XMPP Dev" />
        <item jid="chat.example.com"
              node="slixdev"
              name="Slixmpp Dev" />
    </query>
</iq>

```

Stanza Interface:

```

node -- The name of the node to either
      query or return info from.
items -- A list of 3-tuples, where each tuple contains
        the JID, node, and name of an item.

```

`add_item(jid, node=None, name=None)`

Add a new item element. Each item is required to have a JID, but may also specify a node value to reference non-addressable entities.

Parameters

- **jid** – The JID for the item.
- **node** – Optional additional information to reference non-addressable items.
- **name** – Optional human readable name for the item.

`del_item(jid, node=None)`

Remove a single item.

Parameters

- **jid** – JID of the item to remove.
- **node** – Optional extra identifying information.

`del_items()`

Remove all items.

`get_items()`

Return all items.

```
interfaces = {'items', 'node'}
```

```
name = 'query'
```

```
namespace = 'http://jabber.org/protocol/disco#items'
```

```
plugin_attrib = 'disco_items'
```

```
plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0030.stanza.items.DiscoItem'>}
```

```
plugin_iterables = {<class 'slixmpp.plugins.xep_0030.stanza.items.DiscoItem'>}
```

```
plugin_overrides = {}
```

```
plugin_tag_map = {'{http://jabber.org/protocol/disco#items}item': <class 'slixmpp.plugins.xep_0030.stanza.items.DiscoItem'>}
```

set_items (*items*)

Set or replace all items. The given items must be in a list or set where each item is a tuple of the form:
(jid, node, name)

Parameters **items** – A series of items in tuple format.

setup (*xml=None*)

Populate the stanza object using an optional XML object.

Overrides ElementBase.setup

Caches item information.

Parameters **xml** – Use an existing XML object for the stanza's values.

XEP 0033

```
class slixmpp.plugins.xep_0033.XEP_0033(xmpp, config=None)
```

XEP-0033: Extended Stanza Addressing

```
stanza = <module 'slixmpp.plugins.xep_0033.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0033.stanza.Address(xml=None, parent=None)
```

```
address_types = {'bcc', 'cc', 'noreply', 'replyroom', 'replyto', 'to'}
```

```
get_delivered()
```

```
get_jid()
```

```
interfaces = {'delivered', 'desc', 'jid', 'node', 'type', 'uri'}
```

```
name = 'address'
```

```
namespace = 'http://jabber.org/protocol/address'
```

```
plugin_attrib = 'address'
```

```
set_delivered(delivered)
```

```
set_jid(value)
```

```
set_uri(uri)
```

```
class slixmpp.plugins.xep_0033.stanza.Addresses(xml=None, parent=None)
```

```
add_address(atype='to', jid='', node='', uri='', desc='', delivered=False)
```

```
del_addresses()
```

```
del_all()
```

```
del_bcc()
```

```
del_cc()
```

```

del_noreply()
del_replyroom()
del_replyto()
del_to()
get_addresses()
get_all()
get_bcc()
get_cc()
get_noreply()
get_replyroom()
get_replyto()
get_to()

interfaces = {'addresses', 'all', 'bcc', 'cc', 'noreply', 'replyroom', 'replyto', 'to'}
name = 'addresses'
namespace = 'http://jabber.org/protocol/address'
plugin_attrib = 'addresses'
plugin_attrib_map = {'address': <class 'slixmpp.plugins.xep_0033.stanza.Address'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0033.stanza.Address'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/address}address': <class 'slixmpp.plugin.Address'>}
set_addresses(value)
set_all(value)
set_bcc(value)
set_cc(value)
set_noreply(value)
set_replyroom(value)
set_replyto(value)
set_to(value)

slixmpp.plugins.xep_0033.stanza.del_multi(self)
slixmpp.plugins.xep_0033.stanza.get_multi(self)
slixmpp.plugins.xep_0033.stanza.set_multi(self, value)

```

XEP 0045

```
class slixmpp.plugins.xep_0045.XEP_0045(xmpp, config=None)
    Implements XEP-0045 Multi-User Chat

    async cancel_config(room, *, ifrom=None, **iqkwargs)
        Cancel a requested config form

        Return type Iq

    client_handle_presence(pr)
        As a client, handle a presence stanza

    decline(room, jid, reason='', *, mfrom=None)
        Decline a mediated invitation.

    async destroy(room, reason='', altroom='', *, ifrom=None, **iqkwargs)
        Destroy a room.

    async get_affiliation_list(room, affiliation, *, ifrom=None, **iqkwargs)
        "Get a list of JIDs with the specified affiliation

        Return type List[JID]

    get_jid_property(room, nick, jid_property)
        Get the property of a nick in a room, such as its 'jid' or 'affiliation'. If not found, return None.

    get_our_jid_in_room(room_jid)
        Return the jid we're using in a room.

        Return type str

    async get_roles_list(room, role, *, ifrom=None, **iqkwargs)
        "Get a list of JIDs with the specified role

        Return type List[str]

    async get_room_config(room, ifrom='')
        Get the room config form in 0004 plugin format

    get_roster(room)
        Get the list of nicks in a room.

        Return type List[str]

    handle_config_change(msg)
        Handle a MUC configuration change (with status code).

    handle_groupchat_decline(decl)
        Handle an invitation decline.

    handle_groupchat_error_message(msg)
        Handle a message error event in a muc.

    handle_groupchat_invite(inv)
        Handle an invite into a muc.

    handle_groupchat_join(pr)
        Received a join presence (as a component)

    handle_groupchat_message(msg)
        Handle a message event in a muc.

        Return type None
```

```
handle_groupchat_presence(pr)
    Handle a presence in a muc.

handle_groupchat_subject(msg)
    Handle a message coming from a muc indicating a change of subject (or announcing it when joining the room)

    Return type None

invite(room, jid, reason='', *, mfrom=None)
    Invite a jid to a room.

join_muc(room, nick, maxhistory='0', password='', pstatus='', pshow='', pfrom='')
    Join the specified room, requesting 'maxhistory' lines of history.

leave_muc(room, nick, msg='', pfrom=None)
    Leave the specified room.

async send_affiliation_list(room, affiliations, *, ifrom=None, **iqkwargs)
    Send an affiliation delta list

    Return type Iq

async send_role_list(room, roles, *, ifrom=None, **iqkwargs)
    Send a role delta list

    Return type Iq

async set_affiliation(room, affiliation, *, jid=None, nick=None, reason='', ifrom=None, **iqkwargs)
    Change room affiliation.

async set_role(room, nick, role, *, reason='', ifrom=None, **iqkwargs)
    Change role property of a nick in a room. Typically, roles are temporary (they last only as long as you are in the room), whereas affiliations are permanent (they last across groupchat sessions).

async set_room_config(room, config, *, ifrom=None, **iqkwargs)
    Send a room config form

    Return type Iq

set_subject(room, subject, *, mfrom=None)
    Set a room's subject.

stanza = <module 'slixmpp.plugins.xep_0045.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz Copyright (C) 2020 “Maxime “pep” Buquet <pep@bouah.net>” This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0045.stanza.MUCActor(xml=None, parent=None)
```

```
get_jid()

    Return type Optional[JID]

interfaces = {'jid', 'nick'}
name = 'actor'
```

```
namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'actor'

class slixmpp.plugins.xep_0045.stanza.MUCAdminItem(xml=None, parent=None)

    interfaces = {'affiliation', 'jid', 'nick', 'reason', 'role'}
    name = 'item'
    namespace = 'http://jabber.org/protocol/muc#admin'
    plugin_attrib = 'item'
    sub_interfaces = {'reason'}

class slixmpp.plugins.xep_0045.stanza.MUCAdminQuery(xml=None, parent=None)

    name = 'query'
    namespace = 'http://jabber.org/protocol/muc#admin'
    plugin_attrib = 'mucadmin_query'

class slixmpp.plugins.xep_0045.stanza.MUCBase(xml=None, parent=None)

    del_affiliation()
    del_item_attr(attr)
    del_jid()
    del_nick()
    del_role()
    del_room()
    del_status_codes()
    get_affiliation()
    get_item_attr(attr, default)
    get_jid()

        Return type JID

    get_nick()

        Return type str

    get_role()

        Return type str

    get_room()

        Return type str

    get_status_codes()

        Return type Set[str]

    interfaces = {'affiliation', 'jid', 'nick', 'role', 'room', 'status_codes'}
    name = 'x'
```

```

namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'muc'
set_affiliation(value)
set_item_attr(attr, value)
set_jid(value)
set_nick(value)
set_role(value)
set_room(value)
set_status_codes(codes)

class slixmpp.plugins.xep_0045.stanza.MUCDecline(xml=None, parent=None)

interfaces = {'from', 'reason', 'to'}
name = 'decline'
namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'decline'
sub_interfaces = {'reason'}

class slixmpp.plugins.xep_0045.stanza.MUCHistory(xml=None, parent=None)

interfaces = {'maxchars', 'maxstanzas', 'seconds', 'since'}
name = 'history'
namespace = 'http://jabber.org/protocol/muc'
plugin_attrib = 'history'

class slixmpp.plugins.xep_0045.stanza.MUCInvite(xml=None, parent=None)

interfaces = {'from', 'reason', 'to'}
name = 'invite'
namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'invite'
sub_interfaces = {'reason'}

class slixmpp.plugins.xep_0045.stanza.MUCJoin(xml=None, parent=None)

interfaces = {'password'}
name = 'x'
namespace = 'http://jabber.org/protocol/muc'
plugin_attrib = 'muc_join'
sub_interfaces = {'password'}

class slixmpp.plugins.xep_0045.stanza.MUCMessage(xml=None, parent=None)
A MUC Message

```

```
<message from='foo@muc/user1' type='groupchat' id='someid'>
    <body>Foo</body>
    <x xmlns='http://jabber.org/protocol/muc#user'>
        <item affiliation='none'
              role='none'
              nick='newnick2'
              jid='some@jid'/>
    </x>
</message>
```

```
class slixmpp.plugins.xep_0045.stanza.MUCOwnerDestroy (xml=None, parent=None)
```

```
interfaces = {'jid', 'reason'}
name = 'destroy'
plugin_attrib = 'destroy'
sub_interfaces = {'reason'}
```

```
class slixmpp.plugins.xep_0045.stanza.MUCOwnerQuery (xml=None, parent=None)
```

```
name = 'query'
namespace = 'http://jabber.org/protocol/muc#owner'
plugin_attrib = 'mucowner_query'
```

```
class slixmpp.plugins.xep_0045.stanza.MUCPresence (xml=None, parent=None)
```

A MUC Presence

```
<presence from='foo@muc/user1' type='unavailable'>
    <x xmlns='http://jabber.org/protocol/muc#user'>
        <item affiliation='none'
              role='none'
              nick='newnick2'
              jid='some@jid'/>
        <status code='303' />
    </x>
</presence>
```

```
class slixmpp.plugins.xep_0045.stanza.MUCStatus (xml=None, parent=None)
```

```
interfaces = {'code'}
name = 'status'
namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'status'
set_code (code)
```

```
class slixmpp.plugins.xep_0045.stanza.MUCUserItem (xml=None, parent=None)
```

```
get_jid()
```

Return type Optional[*JID*]

```
interfaces = {'affiliation', 'jid', 'nick', 'reason', 'role'}
```

```

name = 'item'
namespace = 'http://jabber.org/protocol/muc#user'
plugin_attrib = 'item'
sub_interfaces = {'reason'}

```

XEP 0047

```
class slixmpp.plugins.xep_0047.XEP_0047(xmpp, config=None)
```

Stanza elements

```
class slixmpp.plugins.xep_0047.stanza.Close(xml=None, parent=None)
```

```

interfaces = {'sid'}
name = 'close'
namespace = 'http://jabber.org/protocol/ibb'
plugin_attrib = 'ibb_close'

class slixmpp.plugins.xep_0047.stanza.Data(xml=None, parent=None)

```

```

del_data()
get_data()
get_seq()
interfaces = {'data', 'seq', 'sid'}
name = 'data'
namespace = 'http://jabber.org/protocol/ibb'
plugin_attrib = 'ibb_data'
set_data(value)
set_seq(value)
sub_interfaces = {'data'}

```

```
class slixmpp.plugins.xep_0047.stanza.Open(xml=None, parent=None)
```

```

del_block_size()
get_block_size()
interfaces = {'block_size', 'sid', 'stanza'}
name = 'open'
namespace = 'http://jabber.org/protocol/ibb'
plugin_attrib = 'ibb_open'
set_block_size(value)

```

```
slixmpp.plugins.xep_0047.stanza.from_b64(data)
```

```
slixmpp.plugins.xep_0047.stanza.to_b64(data)
```

XEP 0049

```
class slixmpp.plugins.xep_0049.XEP_0049(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0049.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0049.stanza.PrivateXML(xml=None, parent=None)
```

```
interfaces = {}
name = 'query'
namespace = 'jabber:iq:private'
plugin_attrib = 'private'
```

XEP 0050

```
class slixmpp.plugins.xep_0050.XEP_0050(xmpp, config=None)
```

XEP-0050: Ad-Hoc Commands

XMPP's Adhoc Commands provides a generic workflow mechanism for interacting with applications. The result is similar to menu selections and multi-step dialogs in normal desktop applications. Clients do not need to know in advance what commands are provided by any particular application or agent. While adhoc commands provide similar functionality to Jabber-RPC, adhoc commands are used primarily for human interaction.

Also see <<http://xmpp.org/extensions/xep-0050.html>>

Events: command_execute – Received a command with action=”execute” command_next – Received a command with action=”next” command_complete – Received a command with action=”complete” command_cancel – Received a command with action=”cancel”

Attributes:

commands – A dictionary mapping JID/node pairs to command names and handlers.

sessions – A dictionary or equivalent backend mapping session IDs to dictionaries containing data relevant to a command’s session.

```
add_command(jid=None, node=None, name='', handler=None)
```

Make a new command available to external entities.

Access control may be implemented in the provided handler.

Command workflow is done across a sequence of command handlers. The first handler is given the initial Iq stanza of the request in order to support access control. Subsequent handlers are given only the payload items of the command. All handlers will receive the command’s session data.

Parameters

- **jid** – The JID that will expose the command.

- **node** – The node associated with the command.
- **name** – A human readable name for the command.
- **handler** – A function that will generate the response to the initial command request, as well as enforcing any access control policies.

cancel_command(*session*)

Cancel the execution of a command.

Parameters **session** – All stored data relevant to the current command session.

complete_command(*session*)

Finish the execution of a command workflow.

Parameters **session** – All stored data relevant to the current command session.

continue_command(*session, direction='next'*)

Execute the next action of the command.

Parameters **session** – All stored data relevant to the current command session.

get_commands(*jid, **kwargs*)

Return a list of commands provided by a given JID.

Parameters

- **jid** – The JID to query for commands.
- **local** – If true, then the query is for a JID/node combination handled by this Slixmpp instance and no stanzas need to be sent. Otherwise, a disco stanza must be sent to the remove JID to retrieve the items.
- **iterator** – If True, return a result set iterator using the XEP-0059 plugin, if the plugin is loaded. Otherwise the parameter is ignored.

new_session()

Return a new session ID.

prep_handlers(*handlers, **kwargs*)

Prepare a list of functions for use by the backend service.

Intended to be replaced by the backend service as needed.

Parameters

- **handlers** – A list of function pointers
- **kwargs** – Any additional parameters required by the backend.

send_command(*jid, node, ifrom=None, action='execute', payload=None, sessionid=None, flow=False, **kwargs*)

Create and send a command stanza, without using the provided workflow management APIs.

Parameters

- **jid** – The JID to send the command request or result.
- **node** – The node for the command.
- **ifrom** – Specify the sender's JID.
- **action** – May be one of: execute, cancel, complete, or cancel.
- **payload** – Either a list of payload items, or a single payload item such as a data form.
- **sessionid** – The current session's ID value.

- **flow** – If True, process the Iq result using the command workflow methods contained in the session instead of returning the response stanza itself. Defaults to False.

`set_backend(db)`

Replace the default session storage dictionary with a generic, external data storage mechanism.

The replacement backend must be able to interact through the same syntax and interfaces as a normal dictionary.

Parameters `db` – The new session storage mechanism.

`stanza = <module 'slixmpp.plugins.xep_0050.stanza' from '/home/docs/checkouts/readthed`

`start_command(jid, node, session, ifrom=None)`

Initiate executing a command provided by a remote agent.

The provided session dictionary should contain:

Parameters

- **next** – A handler for processing the command result.
- **error** – A handler for processing any error stanzas generated by the request.
- **jid** – The JID to send the command request.
- **node** – The node for the desired command.
- **session** – A dictionary of relevant session data.

`terminate_command(session)`

Delete a command's session after a command has completed or an error has occurred.

Parameters `session` – All stored data relevant to the current command session.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

`class slixmpp.plugins.xep_0050.stanza.Command(xml=None, parent=None)`

XMPP's Adhoc Commands provides a generic workflow mechanism for interacting with applications. The result is similar to menu selections and multi-step dialogs in normal desktop applications. Clients do not need to know in advance what commands are provided by any particular application or agent. While adhoc commands provide similar functionality to Jabber-RPC, adhoc commands are used primarily for human interaction.

Also see <<http://xmpp.org/extensions/xep-0050.html>>

Example command stanzas:

```
<iq type="set">
    <command xmlns="http://jabber.org/protocol/commands"
        node="run_foo"
        action="execute" />
</iq>

<iq type="result">
    <command xmlns="http://jabber.org/protocol/commands"
        node="run_foo"
        sessionid="12345"
        status="executing">
```

(continues on next page)

(continued from previous page)

```

<actions>
    <complete />
</actions>
<note type="info">Information!</note>
<x xmlns="jabber:x:data">
    <field var="greeting"
        type="text-single"
        label="Greeting" />
</x>
</command>
</iq>

```

Stanza Interface:

action	— The action to perform.
actions	— The set of allowable next actions.
node	— The node associated with the command.
notes	— A list of tuples for informative notes.
sessionid	— A unique identifier for a command session.
status	— May be one of: canceled, completed, or executing.

actions = {'cancel', 'complete', 'execute', 'next', 'prev'}**add_note**(msg='', ntype='info')

Add a single note annotation to the command.

Arguments: msg – A human readable message. ntype – One of: ‘info’, ‘warning’, ‘error’**del_actions()**

Remove all allowable next actions.

del_notes()

Remove all notes associated with the command result.

get_action()

Return the value of the action attribute.

If the Iq stanza’s type is “set” then use a default value of “execute”.

get_actions()

Return the set of allowable next actions.

get_notes()

Return a list of note information.

Example:

```
[('info', 'Some informative data'), ('warning', 'Use caution'), ('error', 'The command ran, but had errors')]
```

interfaces = {'action', 'actions', 'node', 'notes', 'sessionid', 'status'}**name** = 'command'**namespace** = 'http://jabber.org/protocol/commands'**next_actions** = {'complete', 'next', 'prev'}**plugin_attrib** = 'command'**set_actions**(values)

Assign the set of allowable next actions.

Parameters values – A list containing any combination of: ‘prev’, ‘next’, and ‘complete’

set_notes (notes)

Add multiple notes to the command result.

Each note is a tuple, with the first item being one of: ‘info’, ‘warning’, or ‘error’, and the second item being any human readable message.

Example:

```
[("info", "Some informative data"), ("warning", "Use caution"), ("error", "The command ran, but had errors")]
```

Arguments: notes – A list of tuples of note information.

```
statuses = {'canceled', 'completed', 'executing'}
```

XEP 0054

```
class slixmpp.plugins.xep_0054.XEP_0054(xmpp, config=None)
XEP-0054: vcard-temp
```

Stanza elements

```
class slixmpp.plugins.xep_0054.stanza.Address(xml=None, parent=None)
```

```
bool_interfaces = {'DOM', 'HOME', 'INTL', 'PREF', 'WORK'}
interfaces = {'CTRY', 'DOM', 'EXTADD', 'HOME', 'INTL', 'LOCALITY', 'PARCEL', 'PCODE',
name = 'ADR'
namespace = 'vcard-temp'
plugin_attrib = 'ADR'
plugin_multi_attrib = 'addresses'
sub_interfaces = {'CTRY', 'EXTADD', 'LOCALITY', 'PCODE', 'POBOX', 'REGION', 'STREET'}
```

```
class slixmpp.plugins.xep_0054.stanza.Agent(xml=None, parent=None)
```

```
interfaces = {'EXTVAL'}
name = 'AGENT'
namespace = 'vcard-temp'
plugin_attrib = 'AGENT'
plugin_attrib_map = {'vcard_temp': <class 'slixmpp.plugins.xep_0054.stanza.VCardTemp'>
plugin_iterables = {}
plugin_multi_attrib = 'agents'
plugin_overrides = {}
plugin_tag_map = {'{vcard-temp}vCard': <class 'slixmpp.plugins.xep_0054.stanza.VCardT
sub_interfaces = {'EXTVAL'}
```

```
class slixmpp.plugins.xep_0054.stanza.BinVal (xml=None, parent=None)
```

```
    del_binval()
    get_binval()
    interfaces = {'BINVAL'}
    is_extension = True
    name = 'BINVAL'
    namespace = 'vcard-temp'
    plugin_attrib = 'BINVAL'
    set_binval (value)
    setup (xml=None)
```

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters **xml** – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0054.stanza.Birthday (xml=None, parent=None)
```

```
    get_bday()
    interfaces = {'BDAY'}
    is_extension = True
    name = 'BDAY'
    namespace = 'vcard-temp'
    plugin_attrib = 'BDAY'
    plugin_multi_attrib = 'birthdays'
    set_bday (value)
```

```
class slixmpp.plugins.xep_0054.stanza.Categories (xml=None, parent=None)
```

```
    del_categories()
    get_categories()
    interfaces = {'CATEGORIES'}
    is_extension = True
    name = 'CATEGORIES'
    namespace = 'vcard-temp'
    plugin_attrib = 'CATEGORIES'
    plugin_multi_attrib = 'categories'
    set_categories (values)
```

```
class slixmpp.plugins.xep_0054.stanza.Classification(xml=None, parent=None)

    bool_interfaces = {'CONFIDENTIAL', 'PRIVATE', 'PUBLIC'}
    interfaces = {'CONFIDENTIAL', 'PRIVATE', 'PUBLIC'}
    name = 'CLASS'
    namespace = 'vcard-temp'
    plugin_attrib = 'CLASS'
    plugin_multi_attrib = 'classifications'

class slixmpp.plugins.xep_0054.stanza.Desc(xml=None, parent=None)

    get_desc()
    interfaces = {'DESC'}
    is_extension = True
    name = 'DESC'
    namespace = 'vcard-temp'
    plugin_attrib = 'DESC'
    plugin_multi_attrib = 'descriptions'
    set_desc(value)

class slixmpp.plugins.xep_0054.stanza.Email(xml=None, parent=None)

    bool_interfaces = {'HOME', 'INTERNET', 'PREF', 'WORK', 'X400'}
    interfaces = {'HOME', 'INTERNET', 'PREF', 'USERID', 'WORK', 'X400'}
    name = 'EMAIL'
    namespace = 'vcard-temp'
    plugin_attrib = 'EMAIL'
    plugin_multi_attrib = 'emails'
    sub_interfaces = {'USERID'}

class slixmpp.plugins.xep_0054.stanza.Geo(xml=None, parent=None)

    interfaces = {'LAT', 'LON'}
    name = 'GEO'
    namespace = 'vcard-temp'
    plugin_attrib = 'GEO'
    plugin_multi_attrib = 'geolocations'
    sub_interfaces = {'LAT', 'LON'}

class slixmpp.plugins.xep_0054.stanza.JabberID(xml=None, parent=None)

    get_jabberid()
```

```

interfaces = {'JABBERID'}
is_extension = True
name = 'JABBERID'
namespace = 'vcard-temp'
plugin_attrib = 'JABBERID'
plugin_multi_attrib = 'jids'
set_jabberid(value)

class slixmpp.plugins.xep_0054.stanza.Label(xml=None, parent=None)

    add_line(value)
    bool_interfaces = {'DOM', 'HOME', 'INT', 'PARCEL', 'POSTAL', 'PREF', 'WORK'}
    del_lines()
    get_lines()
    interfaces = {'DOM', 'HOME', 'INT', 'PARCEL', 'POSTAL', 'PREF', 'WORK', 'lines'}
    name = 'LABEL'
    namespace = 'vcard-temp'
    plugin_attrib = 'LABEL'
    plugin_multi_attrib = 'labels'
    set_lines(values)

class slixmpp.plugins.xep_0054.stanza.Logo(xml=None, parent=None)

    interfaces = {'EXTVAL', 'TYPE'}
    name = 'LOGO'
    namespace = 'vcard-temp'
    plugin_attrib = 'LOGO'
    plugin_attrib_map = {'BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
    plugin_iterables = {}
    plugin_multi_attrib = 'logos'
    plugin_overrides = {}
    plugin_tag_map = {'{vcard-temp}BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
    sub_interfaces = {'EXTVAL', 'TYPE'}

class slixmpp.plugins.xep_0054.stanza.Mailer(xml=None, parent=None)

    get_mailer()
    interfaces = {'MAILER'}
    is_extension = True
    name = 'MAILER'

```

```
namespace = 'vcard-temp'
plugin_attrib = 'MAILER'
plugin_multi_attrib = 'mailers'
set_mailer(value)

class slixmpp.plugins.xep_0054.stanza.Name(xml=None, parent=None)

    get_family()
    get_given()
    get_middle()
    get_prefix()
    get_suffix()

    interfaces = {'FAMILY', 'GIVEN', 'MIDDLE', 'PREFIX', 'SUFFIX'}
    name = 'N'
    namespace = 'vcard-temp'
    plugin_attrib = 'N'
    set_family(value)
    set_given(value)
    set_middle(value)
    set_prefix(value)
    set_suffix(value)

    sub_interfaces = {'FAMILY', 'GIVEN', 'MIDDLE', 'PREFIX', 'SUFFIX'}

class slixmpp.plugins.xep_0054.stanza.Nickname(xml=None, parent=None)

    getNickname()
    interfaces = {'NICKNAME'}
    is_extension = True
    name = 'NICKNAME'
    namespace = 'vcard-temp'
    plugin_attrib = 'NICKNAME'
    plugin_multi_attrib = 'nicknames'
    setNickname(value)

class slixmpp.plugins.xep_0054.stanza.Note(xml=None, parent=None)

    getNote()
    interfaces = {'NOTE'}
    is_extension = True
    name = 'NOTE'
```

```

namespace = 'vcard-temp'
plugin_attrib = 'NOTE'
plugin_multi_attrib = 'notes'
set_note(value)

class slixmpp.plugins.xep_0054.stanza.Org(xml=None, parent=None)

    add_orgunit(value)
    del_orgunits()
    get_orgunits()
    interfaces = {'ORGNAME', 'ORGUNIT', 'orgunits'}
    name = 'ORG'
    namespace = 'vcard-temp'
    plugin_attrib = 'ORG'
    plugin_multi_attrib = 'organizations'
    set_orgunits(values)
    sub_interfaces = {'ORGNAME', 'ORGUNIT'}

class slixmpp.plugins.xep_0054.stanza.Photo(xml=None, parent=None)

    interfaces = {'EXTVAL', 'TYPE'}
    name = 'PHOTO'
    namespace = 'vcard-temp'
    plugin_attrib = 'PHOTO'
    plugin_attrib_map = {'BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
    plugin_iterables = {}
    plugin_multi_attrib = 'photos'
    plugin_overrides = {}
    plugin_tag_map = {'{vcard-temp}BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
    sub_interfaces = {'EXTVAL', 'TYPE'}

class slixmpp.plugins.xep_0054.stanza.ProdID(xml=None, parent=None)

    get_prodid()
    interfaces = {'PRODID'}
    is_extension = True
    name = 'PRODID'
    namespace = 'vcard-temp'
    plugin_attrib = 'PRODID'
    plugin_multi_attrib = 'product_ids'

```

```
    set_prodid(value)

class slixmpp.plugins.xep_0054.stanza.Rev(xml=None, parent=None)

        get_rev()
        interfaces = {'REV'}
        is_extension = True
        name = 'REV'
        namespace = 'vcard-temp'
        plugin_attrib = 'REV'
        plugin_multi_attrib = 'revision_dates'
        set_rev(value)

class slixmpp.plugins.xep_0054.stanza.Role(xml=None, parent=None)

        get_role()
        interfaces = {'ROLE'}
        is_extension = True
        name = 'ROLE'
        namespace = 'vcard-temp'
        plugin_attrib = 'ROLE'
        plugin_multi_attrib = 'roles'
        set_role(value)

class slixmpp.plugins.xep_0054.stanza.SortString(xml=None, parent=None)

        get_sort_string()
        interfaces = {'SORT-STRING'}
        is_extension = True
        name = 'SORT-STRING'
        namespace = 'vcard-temp'
        plugin_attrib = 'SORT_STRING'
        plugin_multi_attrib = 'sort_strings'
        set_sort_string(value)

class slixmpp.plugins.xep_0054.stanza.Sound(xml=None, parent=None)

        interfaces = {'EXTVAL', 'PHONETC'}
        name = 'SOUND'
        namespace = 'vcard-temp'
        plugin_attrib = 'SOUND'
```

```

plugin_attrib_map = {'BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
plugin_iterables = {}
plugin_multi_attrib = 'sounds'
plugin_overrides = {}
plugin_tag_map = {'{vcard-temp}BINVAL': <class 'slixmpp.plugins.xep_0054.stanza.BinVal'>}
sub_interfaces = {'EXTVAL', 'PHONETC'}

class slixmpp.plugins.xep_0054.stanza.Telephone (xml=None, parent=None)
    bool_interfaces = {'BBS', 'CELL', 'FAX', 'HOME', 'ISDN', 'MODEM', 'MSG', 'PAGER', 'PCS'}
    del_number()
```

interfaces = {'BBS', 'CELL', 'FAX', 'HOME', 'ISDN', 'MODEM', 'MSG', 'NUMBER', 'PAGER', 'TEL'}

name = 'TEL'

namespace = 'vcard-temp'

plugin_attrib = 'TEL'

plugin_multi_attrib = 'telephone_numbers'

set_number (*value*)

setup (*xml=None*)
 Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters **xml** – An existing XML object to use for the stanza's content instead of generating new XML.

```

    sub_interfaces = {'NUMBER'}
```

class slixmpp.plugins.xep_0054.stanza.TimeZone (*xml=None, parent=None*)
 get_tz()

interfaces = {'TZ'}

is_extension = True

name = 'TZ'

namespace = 'vcard-temp'

plugin_attrib = 'TZ'

plugin_multi_attrib = 'timezones'

set_tz (*value*)

class slixmpp.plugins.xep_0054.stanza.Title (*xml=None, parent=None*)
 get_title()

interfaces = {'TITLE'}

is_extension = True

```
name = 'TITLE'
namespace = 'vcard-temp'
plugin_attrib = 'TITLE'
plugin_multi_attrib = 'titles'
set_title(value)

class slixmpp.plugins.xep_0054.stanza.UID(xml=None, parent=None)

    get_uid()
    interfaces = {'UID'}
    is_extension = True
    name = 'UID'
    namespace = 'vcard-temp'
    plugin_attrib = 'UID'
    plugin_multi_attrib = 'uids'
    set_uid(value)

class slixmpp.plugins.xep_0054.stanza.URL(xml=None, parent=None)

    get_url()
    interfaces = {'URL'}
    is_extension = True
    name = 'URL'
    namespace = 'vcard-temp'
    plugin_attrib = 'URL'
    plugin_multi_attrib = 'urls'
    set_url(value)

class slixmpp.plugins.xep_0054.stanza.VCardTemp(xml=None, parent=None)

    interfaces = {'FN', 'VERSION'}
    name = 'vCard'
    namespace = 'vcard-temp'
    plugin_attrib = 'vcard_temp'
    plugin_attrib_map = {'ADR': <class 'slixmpp.plugins.xep_0054.stanza.Address'>, 'AGENT': <class 'slixmpp.plugins.xep_0054.stanza.Agent'>, 'COUNTRY': <class 'slixmpp.plugins.xep_0054.stanza.Country'>, 'EMAIL': <class 'slixmpp.plugins.xep_0054.stanza.Email'>, 'GENDER': <class 'slixmpp.plugins.xep_0054.stanza.Gender'>, 'IM': <class 'slixmpp.plugins.xep_0054.stanza.Im'>, 'NAME': <class 'slixmpp.plugins.xep_0054.stanza.Name'>, 'NICKNAME': <class 'slixmpp.plugins.xep_0054.stanza.Nickname'>, 'ORGANIZATION': <class 'slixmpp.plugins.xep_0054.stanza.Organization'>, 'PHONE': <class 'slixmpp.plugins.xep_0054.stanza.Phone'>, 'POSTALADDRESS': <class 'slixmpp.plugins.xep_0054.stanza.PostalAddress'>, 'ROLE': <class 'slixmpp.plugins.xep_0054.stanza.Role'>, 'SKYPE': <class 'slixmpp.plugins.xep_0054.stanza.Skype'>, 'TITLE': <class 'slixmpp.plugins.xep_0054.stanza.Title'>, 'WEBSITE': <class 'slixmpp.plugins.xep_0054.stanza.Website'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0054.stanza.Label'>, <class 'slixmpp.plugins.xep_0054.stanza.MultiLabel'>}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi'>}
    sub_interfaces = {'FN', 'VERSION'}
```

XEP 0059

```
class slixmpp.plugins.xep_0059.XEP_0059(xmpp, config=None)
XEP-0050: Result Set Management

iterate(stanza, interface, results='substanzas', amount=10, reverse=False, recv_interface=None,
         pre_cb=None, post_cb=None)
Create a new result set iterator for a given stanza query.
```

Arguments:

stanza – A stanza object to serve as a template for queries made each iteration. For example, a basic disco#items query.

interface – The name of the stanza to which the result set management stanza should be appended in the query stanza. For example, for disco#items queries the interface ‘disco_items’ should be used.

recv_interface – The name of the stanza from which the result set management stanza should be read in the result stanza. If unspecified, it will be set to the same value as the **interface** parameter.

pre_cb – Callback to run before sending each stanza e.g. setting the MAM queryid and starting a stanza collector.

post_cb – Callback to run after receiving each stanza e.g. stopping a MAM stanza collector in order to gather results.

results – The name of the interface containing the query results (typically just ‘substanzas’).

```
stanza = <module 'slixmpp.plugins.xep_0059.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz, Erik Reuterborg Larsson This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0059.stanza.Set(xml=None, parent=None)
XEP-0059 (Result Set Management) can be used to manage the results of queries. For example, limiting the number of items per response or starting at certain positions.
```

Example set stanzas:

```
<iq type="get">
  <query xmlns="http://jabber.org/protocol/disco#items">
    <set xmlns="http://jabber.org/protocol/rsm">
      <max>2</max>
    </set>
  </query>
</iq>

<iq type="result">
  <query xmlns="http://jabber.org/protocol/disco#items">
    <item jid="conference.example.com" />
    <item jid="pubsub.example.com" />
  <set xmlns="http://jabber.org/protocol/rsm">
    <first>conference.example.com</first>
    <last>pubsub.example.com</last>
```

(continues on next page)

(continued from previous page)

```
</set>
</query>
</iq>
```

Stanza Interface:

first_index	-- The index attribute of <first>
after	-- The id defining from which item to start
before	-- The id defining from which item to start when browsing backwards
max	-- Max amount per response
first	-- Id for the first item in the response
last	-- Id for the last item in the response
index	-- Used to set an index to start from
count	-- The number of remote items available

`del_first_index()`

Removes the index attribute for <first> but keeps the element

`get_before()`

Returns the value of <before>, if it is empty it will return True

`get_first_index()`

Returns the value of the index attribute for <first>

```
interfaces = {'after', 'before', 'count', 'first', 'first_index', 'index', 'last', 'max'}
```

```
name = 'set'
```

```
namespace = 'http://jabber.org/protocol/rsm'
```

```
plugin_attrib = 'rsm'
```

`set_before(val)`

Sets the value of <before>, if the value is True then the element will be created without a value

`set_first_index(val)`

Sets the index attribute for <first> and creates the element if it doesn't exist

```
sub_interfaces = {'after', 'before', 'count', 'first', 'index', 'last', 'max'}
```

XEP 0060

```
class slixmpp.plugins.xep_0060.XEP_0060(xmpp, config=None)
XEP-0060 Publish Subscribe
```

```
create_node(jid, node, config=None, ntype=None, ifrom=None, timeout_callback=None, callback=None, timeout=None)
Create and configure a new pubsub node.
```

A server MAY use a different name for the node than the one provided, so be sure to check the result stanza for a server assigned name.

If no configuration form is provided, the node will be created using the server's default configuration. To get the default configuration use `get_node_config()`.

Parameters

- `jid` – The JID of the pubsub service.

- **node** – Optional name of the node to create. If no name is provided, the server MAY generate a node ID for you. The server can also assign a different name than the one you provide; check the result stanza to see if the server assigned a name.
- **config** – Optional XEP-0004 data form of configuration settings.
- **ntype** – The type of node to create. Servers typically default to using ‘leaf’ if no type is provided.

delete_node (*jid, node, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Delete a a pubsub node.

Parameters

- **jid** – The JID of the pubsub service.
- **node** – The node to delete.

get_item (*jid, node, item_id, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Retrieve the content of an individual item.

get_item_ids (*jid, node, ifrom=None, timeout_callback=None, callback=None, timeout=None, iterator=False*)
Retrieve the ItemIDs hosted by a given node, using disco.

get_items (*jid, node, item_ids=None, max_items=None, iterator=False, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Request the contents of a node’s items.

The desired items can be specified, or a query for the last few published items can be used.

Pubsub services may use result set management for nodes with many items, so an iterator can be returned if needed.

get_node_affiliations (*jid, node, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Retrieve the affiliations associated with a given node.

Parameters

- **jid** – The JID of the pubsub service.
- **node** – The node to retrieve affiliations from.

get_node_config (*jid, node=None, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Retrieve the configuration for a node, or the pubsub service’s default configuration for new nodes.

Parameters

- **jid** – The JID of the pubsub service.
- **node** – The node to retrieve the configuration for. If None, the default configuration for new nodes will be requested. Defaults to None.

get_node_subscriptions (*jid, node, ifrom=None, timeout_callback=None, callback=None, timeout=None*)
Retrieve the subscriptions associated with a given node.

Parameters

- **jid** – The JID of the pubsub service.
- **node** – The node to retrieve subscriptions from.

get_nodes (*args, **kwargs)
Discover the nodes provided by a Pubsub service, using disco.

`map_node_event (node, event_name)`

Map node names to events.

When a pubsub event is received for the given node, raise the provided event.

For example:

```
map_node_event('http://jabber.org/protocol/tune',
                'user_tune')
```

will produce the events ‘user_tune_publish’ and ‘user_tune_retract’ when the respective notifications are received from the node ‘<http://jabber.org/protocol/tune>’, among other events.

Parameters

- **node** – The node name to map to an event.
- **event_name** – The name of the event to raise when a notification from the given node is received.

`publish (jid, node, id=None, payload=None, options=None, ifrom=None, timeout_callback=None, callback=None, timeout=None)`

Add a new item to a node, or edit an existing item.

For services that support it, you can use the publish command as an event signal by not including an ID or payload.

When including a payload and you do not provide an ID then the service will generally create an ID for you.

Publish options may be specified, and how those options are processed is left to the service, such as treating the options as preconditions that the node’s settings must match.

Parameters

- **jid** – The JID of the pubsub service.
- **node** – The node to publish the item to.
- **id** – Optionally specify the ID of the item.
- **payload** – The item content to publish.
- **options** – A form of publish options.

`purge (jid, node, ifrom=None, timeout_callback=None, callback=None, timeout=None)`

Remove all items from a node.

`retract (jid, node, id, notify=None, ifrom=None, timeout_callback=None, callback=None, timeout=None)`

Delete a single item from a node.

`stanza = <module 'slixmpp.plugins.xep_0060.stanza' from '/home/docs/checkouts/readthedocs/`

`subscribe (jid, node, bare=True, subscribee=None, options=None, ifrom=None, timeout_callback=None, callback=None, timeout=None)`

Subscribe to updates from a pubsub node.

The rules for determining the JID that is subscribing to the node are: 1. If subscribee is given, use that as provided. 2. If ifrom was given, use the bare or full version based on bare. 3. Otherwise, use self.xmpp.boundjid based on bare.

Parameters

- **jid** – The pubsub service JID.

- **node** – The node to subscribe to.
- **bare** – Indicates if the subscribee is a bare or full JID. Defaults to True for a bare JID.
- **subscribee** – The JID that is subscribing to the node.
- **options** –

```
unsubscribe(jid, node, subid=None, bare=True, subscribee=None, ifrom=None, time-out_callback=None, callback=None, timeout=None)
```

Unsubscribe from updates from a pubsub node.

The rules for determining the JID that is unsubscribing from the node are: 1. If subscribee is given, use that as provided. 2. If ifrom was given, use the bare or full version based on bare. 3. Otherwise, use self.xmpp.boundjid based on bare.

Parameters

- **jid** – The pubsub service JID.
- **node** – The node to unsubscribe from.
- **subid** – The specific subscription, if multiple subscriptions exist for this JID/node combination.
- **bare** – Indicates if the subscribee is a bare or full JID. Defaults to True for a bare JID.
- **subscribee** – The JID that is unsubscribing from the node.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0060.stanza.base.OptionalSetting
```

```
    del_required()
    get_required()
    interfaces = {'required'}
    set_required(value)
```

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation(xml=None, parent=None)
```

```
    get_jid()
    interfaces = {'affiliation', 'jid', 'node'}
    name = 'affiliation'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'affiliation'
    set_jid(value)
```

```
class slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations(xml=None, parent=None)

    interfaces = {'node'}
    name = 'affiliations'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'affiliations'
    plugin_attrib_map = {'affiliation': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation'>}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub}affiliation': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation'>}

class slixmpp.plugins.xep_0060.stanza.pubsub.Configure(xml=None, parent=None)

    getType()
    interfaces = {'node', 'type'}
    name = 'configure'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'configure'
    plugin_attrib_map = {'form': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:x:data}x': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}

class slixmpp.plugins.xep_0060.stanza.pubsub.Create(xml=None, parent=None)

    interfaces = {'node'}
    name = 'create'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'create'

class slixmpp.plugins.xep_0060.stanza.pubsub.Default(xml=None, parent=None)

    get_type()
    interfaces = {'node', 'type'}
    name = 'default'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'default'

class slixmpp.plugins.xep_0060.stanza.pubsub.Item(xml=None, parent=None)

    del_payload()
    get_payload()
```

```

interfaces = {'id', 'payload'}
name = 'item'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'item'
set_payload(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Items(xml=None, parent=None)

interfaces = {'max_items', 'node'}
name = 'items'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'items'
plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub}item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
set_max_items(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Options(*args, **kwargs)

del_options()
get_jid()
get_options()
interfaces = {'jid', 'node', 'options'}
name = 'options'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'options'
set_jid(value)
set_options(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Publish(xml=None, parent=None)

interfaces = {'node'}
name = 'publish'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'publish'
plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub}item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}

```

```
class slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions (xml=None,      par-
ent=None)

    del_publish_options()
    get_publish_options()
    interfaces = {'publish_options'}
    is_extension = True
    name = 'publish-options'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'publish_options'
    set_publish_options(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub (xml=None, parent=None)

    interfaces = {}
    name = 'pubsub'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'pubsub'
    plugin_attrib_map = {'affiliations': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub}affiliations': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations'>}
class slixmpp.plugins.xep_0060.stanza.pubsub.Retract (xml=None, parent=None)

    get_notify()
    interfaces = {'node', 'notify'}
    name = 'retract'
    namespace = 'http://jabber.org/protocol/pubsub'
    plugin_attrib = 'retract'
    plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub}item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Item'>}
    set_notify(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe (xml=None, parent=None)

    get_jid()
    interfaces = {'jid', 'node'}
    name = 'subscribe'
```

```

namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'subscribe'
plugin_attrib_map = {'options': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Options'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub}options': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Options'>
set_jid(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions(xml=None, parent=None)

interfaces = {'required'}
name = 'subscribe-options'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'suboptions'

class slixmpp.plugins.xep_0060.stanza.pubsub.Subscription(xml=None, parent=None)

get_jid()
interfaces = {'jid', 'node', 'subid', 'subscription'}
name = 'subscription'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'subscription'
plugin_attrib_map = {'suboptions': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Subscription'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub}subscribe-options': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions'>
set_jid(value)

class slixmpp.plugins.xep_0060.stanza.pubsub.Subscriptions(xml=None, parent=None)

interfaces = {'node'}
name = 'subscriptions'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'subscriptions'
plugin_attrib_map = {'subscription': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Subscription'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Subscription'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub}subscription': <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Subscription'>

class slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe(xml=None, parent=None)

```

```
get_jid()
interfaces = {'jid', 'node', 'subid'}
name = 'unsubscribe'
namespace = 'http://jabber.org/protocol/pubsub'
plugin_attrib = 'unsubscribe'
set_jid(value)
```

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCondition(xml=None,
                                                                parent=None)
                                                               
condition_ns = 'http://jabber.org/protocol/pubsub#errors'
conditions = {'closed-node', 'configuration-required', 'invalid-jid', 'invalid-options'}
del_condition()
    Remove the condition element.

del_unsupported()
    Delete an unsupported feature condition.

get_condition()
    Return the condition element's name.

get_unsupported()
    Return the name of an unsupported feature

interfaces = {'condition', 'unsupported'}
plugin_attrib = 'pubsub'
plugin_attrib_map = {}
plugin_tag_map = {}

set_condition(value)
    Set the tag name of the condition element.

    Arguments: value – The tag name of the condition element.

set_unsupported(value)
    Mark a feature as unsupported

setup(xml)
    Don't create XML for the plugin.
```

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0060.stanza.pubsub_owner.DefaultConfig(*args,
                                                               **kwargs)

get_config()
interfaces = {'config', 'node'}
name = 'default'
```

```

namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib = 'default'
plugin_attrib_map = {'form': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{jabber:x:data}x': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
set_config(value)

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliation(xml=None,
                                                                parent=None)
                                                                ent=None)

interfaces = {'affiliation', 'jid'}
namespace = 'http://jabber.org/protocol/pubsub#owner'

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliations(xml=None,
                                                                parent=None)
                                                                ent=None)

append(affiliation)
Append either an XML object or a stanza to this stanza object.

If a stanza object is appended, it will be added to the list of iterable stanzas.

Allows stanza objects to be used like lists.

Parameters item – Either an XML object or a stanza object to add to this stanza's contents.

interfaces = {'node'}
namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib_map = {'affiliation': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliation'>,
plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation'>, <class 'slixmpp.plugins.xep_0060.stanza.pubsub.OwnerConfigure'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub#owner}affiliation': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConfigure'>}

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConfigure(xml=None,
                                                                parent=None)
                                                                ent=None)

interfaces = {'node'}
name = 'configure'
namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib = 'configure'
plugin_attrib_map = {'form': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{jabber:x:data}x': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>'}

```

```
class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefault(xml=None,
                                                               parent=None)

    interfaces = {'node'}
    namespace = 'http://jabber.org/protocol/pubsub#owner'
    plugin_attrib_map = {'form': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:x:data}x': <class 'slixmpp.plugins.xep_0004.stanza.form.Fo

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDelete(xml=None, par-
                                                               ent=None)

    interfaces = {'node'}
    name = 'delete'
    namespace = 'http://jabber.org/protocol/pubsub#owner'
    plugin_attrib = 'delete'
    plugin_attrib_map = {'redirect': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub#owner}redirect': <class 'slixmp

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerPurge(xml=None, par-
                                                               ent=None)

    interfaces = {'node'}
    name = 'purge'
    namespace = 'http://jabber.org/protocol/pubsub#owner'
    plugin_attrib = 'purge'

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedirect(xml=None,
                                                               parent=None)

    get_jid()
    interfaces = {'jid', 'node'}
    name = 'redirect'
    namespace = 'http://jabber.org/protocol/pubsub#owner'
    plugin_attrib = 'redirect'
    set_jid(value)

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscription(xml=None,
                                                               parent=None)

    get_jid()
    interfaces = {'jid', 'subscription'}
```

```

name = 'subscription'
namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib = 'subscription'
set_jid(value)

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscriptions (xml=None,
                                                                parent=None)
                                                                parent=None)

append(subscription)
Append either an XML object or a stanza to this stanza object.

If a stanza object is appended, it will be added to the list of iterable stanzas.

Allows stanza objects to be used like lists.

    Parameters item – Either an XML object or a stanza object to add to this stanza's contents.

interfaces = {'node'}
name = 'subscriptions'
namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib = 'subscriptions'
plugin_attrib_map = {'subscription': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscriptions'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub.Subscription'>, <class 'slixmpp.plugins.xep_0060.stanza.pubsub.Event'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub#owner}subscription': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscriptions'>,
                  '{http://jabber.org/protocol/pubsub#owner}affiliations': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliations'>}

class slixmpp.plugins.xep_0060.stanza.pubsub_owner.PubsubOwner (xml=None, parent=None)

interfaces = {}
name = 'pubsub'
namespace = 'http://jabber.org/protocol/pubsub#owner'
plugin_attrib = 'pubsub_owner'
plugin_attrib_map = {'affiliations': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliations'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub#owner}affiliations': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliations'>,
                  '{http://jabber.org/protocol/pubsub#owner}subscription': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscriptions'>,
                  '{http://jabber.org/protocol/pubsub#owner}event': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Event'>}

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

class slixmpp.plugins.xep_0060.stanza.pubsub_event.Event (xml=None, parent=None)

interfaces = {}
name = 'event'
namespace = 'http://jabber.org/protocol/pubsub#event'

```

```
plugin_attrib = 'pubsub_event'
plugin_attrib_map = {'collection': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Collection'>}
plugin_iterables = {}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/pubsub#event}collection': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Collection'>,
                  '{http://jabber.org/protocol/pubsub#event}associate': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Associate'>,
                  '{http://jabber.org/protocol/pubsub#event}configuration': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Configuration'>,
                  '{http://jabber.org/protocol/pubsub#event}delete': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Delete'>,
                  '{http://jabber.org/protocol/pubsub#event}event': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Event'>,
                  '{http://jabber.org/protocol/pubsub#event}info': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Info'>,
                  '{http://jabber.org/protocol/pubsub#event}subscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Subscribe'>,
                  '{http://jabber.org/protocol/pubsub#event}unsubscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Unsubscribe'>}
class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventAssociate(xml=None,
                                                                parent=None)
    interfaces = {'node'}
    name = 'associate'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'associate'

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCollection(xml=None,
                                                                parent=None)
    interfaces = {'node'}
    name = 'collection'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'collection'
    plugin_attrib_map = {'associate': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Associate'>,
                         'collection': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Collection'>,
                         'configuration': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Configuration'>,
                         'delete': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Delete'>,
                         'event': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Event'>,
                         'info': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Info'>,
                         'subscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Subscribe'>,
                         'unsubscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Unsubscribe'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub#event}associate': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Associate'>,
                      '{http://jabber.org/protocol/pubsub#event}collection': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Collection'>,
                      '{http://jabber.org/protocol/pubsub#event}configuration': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Configuration'>,
                      '{http://jabber.org/protocol/pubsub#event}delete': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Delete'>,
                      '{http://jabber.org/protocol/pubsub#event}event': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Event'>,
                      '{http://jabber.org/protocol/pubsub#event}info': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Info'>,
                      '{http://jabber.org/protocol/pubsub#event}subscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Subscribe'>,
                      '{http://jabber.org/protocol/pubsub#event}unsubscribe': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.Unsubscribe'>}
class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventConfiguration(xml=None,
                                                                parent=None)
    interfaces = {'node'}
    name = 'configuration'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'configuration'
    plugin_attrib_map = {'form': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:x:data}x': <class 'slixmpp.plugins.xep_0004.stanza.form.Form'>}
class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDelete(xml=None, parent=None)
    del_redirect()
    get_redirect()
```

```

interfaces = {'node', 'redirect'}
name = 'delete'
namespace = 'http://jabber.org/protocol/pubsub#event'
plugin_attrib = 'delete'
set_redirect(uri)

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDisassociate(xml=None,
                                                                parent=None)
    interfaces = {'node'}
    name = 'disassociate'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'disassociate'

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItem(xml=None, parent=None)
    del_payload()
    get_payload()
    interfaces = {'id', 'node', 'payload', 'publisher'}
    name = 'item'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'item'
    set_payload(value)

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItems(xml=None, parent=None)
    interfaces = {'node'}
    name = 'items'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'items'
    plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItem'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.EventRetract'>}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/pubsub#event}item': <class 'slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItem'>}

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventPurge(xml=None, parent=None)
    interfaces = {'node'}
    name = 'purge'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'purge'

```

```
class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventRetract(xml=None,
                                                               parent=None)

    interfaces = {'id'}
    name = 'retract'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'retract'

class slixmpp.plugins.xep_0060.stanza.pubsub_event.EventSubscription(xml=None,
                                                               parent=None)

    get_expiry()
    get_jid()
    interfaces = {'expiry', 'jid', 'node', 'subid', 'subscription'}
    name = 'subscription'
    namespace = 'http://jabber.org/protocol/pubsub#event'
    plugin_attrib = 'subscription'
    set_expiry(value)
    set_jid(value)
```

XEP 0065

```
class slixmpp.plugins.xep_0065.XEP_0065(xmpp, config=None)

    activate(proxy, sid, target, ifrom=None, timeout=None, callback=None)
        Activate the socks5 session that has been negotiated.

    close()
        Closes all proxy sockets.

    deactivate(sid)
        Closes the proxy socket associated with this SID.

    async discover_proxies(jid=None, ifrom=None, timeout=None)
        Auto-discover the JIDs of SOCKS5 proxies on an XMPP server.

    get_network_address(proxy, ifrom=None, timeout=None, callback=None)
        Get the network information of a proxy.

    get_socket(sid)
        Returns the socket associated to the SID.

    async handshake(to, ifrom=None, sid=None, timeout=None)
        Starts the handshake to establish the socks5 bytestreams connection.
```

Stanza elements

```

class slixmpp.plugins.xep_0065.stanza.Socks5 (xml=None, parent=None)
    add_streamhost (jid, host, port)
    interfaces = {'activate', 'sid'}
    name = 'query'
    namespace = 'http://jabber.org/protocol/bytestreams'
    plugin_attrib = 'socks'
    plugin_attrib_map = {'streamhost': <class 'slixmpp.plugins.xep_0065.stanza.StreamHost'
    plugin_iterables = {<class 'slixmpp.plugins.xep_0065.stanza.StreamHost'>}
    plugin_overrides = {}
    plugin_tag_map = {'{http://jabber.org/protocol/bytestreams}streamhost': <class 'slixmp...
    sub_interfaces = {'activate'}

class slixmpp.plugins.xep_0065.stanza.StreamHost (xml=None, parent=None)
    get_jid()
    interfaces = {'host', 'jid', 'port'}
    name = 'streamhost'
    namespace = 'http://jabber.org/protocol/bytestreams'
    plugin_attrib = 'streamhost'
    plugin_multi_attrib = 'streamhosts'
    set_jid (value)
class slixmpp.plugins.xep_0065.stanza.StreamHostUsed (xml=None, parent=None)
    get_jid()
    interfaces = {'jid'}
    name = 'streamhost-used'
    namespace = 'http://jabber.org/protocol/bytestreams'
    plugin_attrib = 'streamhost_used'
    set_jid (value)

```

XEP 0066

```
class slixmpp.plugins.xep_0066.XEP_0066(xmpp, config=None)
XEP-0066: Out of Band Data
```

Out of Band Data is a basic method for transferring files between XMPP agents. The URL of the resource in question is sent to the receiving entity, which then downloads the resource before responding to the OOB request. OOB is also used as a generic means to transmit URLs in other stanzas to indicate where to find additional information.

Also see <<http://www.xmpp.org/extensions/xep-0066.html>>.

Events:

oob_transfer – Raised when a request to download a resource has been received.

```
register_url_handler(jid=None, handler=None)
```

Register a handler to process download requests, either for all JIDs or a single JID.

Parameters

- **jid** – If None, then set the handler as a global default.
- **handler** – If None, then remove the existing handler for the given JID, or reset the global handler if the JID is None.

```
send_oob(to, url, desc=None, ifrom=None, **iqargs)
```

Initiate a basic file transfer by sending the URL of a file or other resource.

Parameters

- **url** – The URL of the resource to transfer.
- **desc** – An optional human readable description of the item that is to be transferred.

```
stanza = <module 'slixmpp.plugins.xep_0066.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0066.stanza.OOB(xml=None, parent=None)
```

```
interfaces = {'desc', 'url'}
name = 'x'
namespace = 'jabber:x:oob'
plugin_attrib = 'oob'
sub_interfaces = {'desc', 'url'}
```

```
class slixmpp.plugins.xep_0066.stanza.OOBTransfer(xml=None, parent=None)
```

```
interfaces = {'desc', 'sid', 'url'}
name = 'query'
namespace = 'jabber:iq:oob'
```

```
plugin_attrib = 'oob_transfer'  
sub_interfaces = {'desc', 'url'}
```

XEP 0070

```
class slixmpp.plugins.xep_0070.XEP_0070(xmpp, config=None)  
    XEP-0070 Verifying HTTP Requests via XMPP  
  
    stanza = <module 'slixmpp.plugins.xep_0070.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2015 Emmanuel Gil Peyrot This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0070.stanza.Confirm(xml=None, parent=None)  
  
    interfaces = {'id', 'method', 'url'}  
    name = 'confirm'  
    namespace = 'http://jabber.org/protocol/http-auth'  
    plugin_attrib = 'confirm'
```

XEP 0071

```
class slixmpp.plugins.xep_0071.XEP_0071(xmpp, config=None)  
  
    stanza = <module 'slixmpp.plugins.xep_0071.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0071.stanza.XHTML_IM(xml=None, parent=None)  
  
    def _body(lang=None):  
        get_body(lang=None)  
            Return the contents of the HTML body.  
        interfaces = {'body'}  
        lang_interfaces = {'body'}  
        name = 'html'  
        namespace = 'http://jabber.org/protocol/xhtml-im'  
        plugin_attrib = 'html'  
        set_body(content, lang=None)
```

XEP 0077

```
class slixmpp.plugins.xep_0077.XEP_0077(xmpp, config=None)
XEP-0077: In-Band Registration
```

```
stanza = <module 'slixmpp.plugins.xep_0077.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0077.stanza.Register(xml=None, parent=None)
```

```
add_field(value)
del_fields()

form_fields = {'address', 'city', 'date', 'email', 'first', 'key', 'last', 'misc', 'name', 'nick', 'remove', 'registered', 'remove', 'interfaces', 'sub_interfaces'}
get_fields()
get_registered()
get_remove()
interfaces = {'address', 'city', 'date', 'email', 'fields', 'first', 'instructions', 'name', 'nick', 'remove', 'sub_interfaces'}
name = 'query'
namespace = 'jabber:iq:register'
plugin_attrib = 'register'
set_fields(fields)
set_registered(value)
set_remove(value)
sub_interfaces = {'address', 'city', 'date', 'email', 'fields', 'first', 'instructions', 'name', 'nick', 'remove', 'sub_interfaces'}
class slixmpp.plugins.xep_0077.stanza.RegisterFeature(xml=None, parent=None)

interfaces = {}
name = 'register'
namespace = 'http://jabber.org/features/iq-register'
plugin_attrib = 'register'
```

XEP 0079

```
class slixmpp.plugins.xep_0079.XEP_0079(xmpp, config=None)
XEP-0079 Advanced Message Processing
```

```
stanza = <module 'slixmpp.plugins.xep_0079.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2013 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0079.stanza.AMP(xml=None, parent=None)
```

```
add_rule(action, condition, value)
del_per_hop()
get_from()
get_per_hop()
get_to()

interfaces = {'from', 'per_hop', 'status', 'to'}
name = 'amp'
namespace = 'http://jabber.org/protocol/amp'
plugin_attrib = 'amp'
plugin_attrib_map = {'rule': <class 'slixmpp.plugins.xep_0079.stanza.Rule'>, 'rules': {}}
plugin_iterables = {<class 'slixmpp.plugins.xep_0079.stanza.Rule'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/amp}rule': <class 'slixmpp.plugins.xep_0079.stanza.Rule'>}
```

```
set_from(value)
set_per_hop(value)
set_to(value)
```

```
class slixmpp.plugins.xep_0079.stanza.AMPFeature(xml=None, parent=None)
```

```
name = 'amp'
namespace = 'http://jabber.org/features/amp'
```

```
class slixmpp.plugins.xep_0079.stanza.FailedRule(xml=None, parent=None)
```

```
namespace = 'http://jabber.org/protocol/amp#errors'
```

```
class slixmpp.plugins.xep_0079.stanza.FailedRules(xml=None, parent=None)
```

```
name = 'failed-rules'
namespace = 'http://jabber.org/protocol/amp#errors'
```

```
plugin_attrib = 'failed_rules'
plugin_attrib_map = {'rule':  <class 'slixmpp.plugins.xep_0079.stanza.FailedRule'>, 'r
plugin_iterables = {<class 'slixmpp.plugins.xep_0079.stanza.FailedRule'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/amp#errors}rule':  <class 'slixmpp.plugins.xep_0079.stanza.InvalidRules'>
class slixmpp.plugins.xep_0079.stanza.InvalidRules(xml=None, parent=None)

name = 'invalid-rules'
namespace = 'http://jabber.org/protocol/amp'
plugin_attrib = 'invalid_rules'
plugin_attrib_map = {'rule':  <class 'slixmpp.plugins.xep_0079.stanza.Rule'>, 'rules': <cl
plugin_iterables = {<class 'slixmpp.plugins.xep_0079.stanza.Rule'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/amp}rule':  <class 'slixmpp.plugins.xep_0079.stanza.Rule'>
class slixmpp.plugins.xep_0079.stanza.Rule(xml=None, parent=None)

interfaces = {'action', 'condition', 'value'}
name = 'rule'
namespace = 'http://jabber.org/protocol/amp'
plugin_attrib = 'rule'
plugin_multi_attrib = 'rules'
class slixmpp.plugins.xep_0079.stanza.UnsupportedActions(xml=None, parent=None)

name = 'unsupported-actions'
namespace = 'http://jabber.org/protocol/amp'
plugin_attrib = 'unsupported_actions'
plugin_attrib_map = {'rule':  <class 'slixmpp.plugins.xep_0079.stanza.Rule'>, 'rules': <cl
plugin_iterables = {<class 'slixmpp.plugins.xep_0079.stanza.Rule'>}
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/amp}rule':  <class 'slixmpp.plugins.xep_0079.stanza.Rule'>
class slixmpp.plugins.xep_0079.stanza.UnsupportedConditions(xml=None, parent=None)

name = 'unsupported-conditions'
namespace = 'http://jabber.org/protocol/amp'
plugin_attrib = 'unsupported_conditions'
plugin_attrib_map = {'rule':  <class 'slixmpp.plugins.xep_0079.stanza.Rule'>, 'rules': <cl
plugin_iterables = {<class 'slixmpp.plugins.xep_0079.stanza.Rule'>}
```

```
plugin_overrides = {}
plugin_tag_map = {'{http://jabber.org/protocol/amp}rule': <class 'slixmpp.plugins.xep_0080.XEP_0080'>}
```

XEP 0080

```
class slixmpp.plugins.xep_0080.XEP_0080(xmpp, config=None)
```

XEP-0080: User Location

```
publish_location(**kwargs)
```

 Publish the user's current location.

Parameters

- **accuracy** – Horizontal GPS error in meters.
- **alt** – Altitude in meters above or below sea level.
- **area** – A named area such as a campus or neighborhood.
- **bearing** – GPS bearing (direction in which the entity is heading to reach its next waypoint), measured in decimal degrees relative to true north.
- **building** – A specific building on a street or in an area.
- **country** – The nation where the user is located.
- **countrycode** – The ISO 3166 two-letter country code.
- **datum** – GPS datum.
- **description** – A natural-language name for or description of the location.
- **error** – Horizontal GPS error in arc minutes. Obsoleted by the accuracy parameter.
- **floor** – A particular floor in a building.
- **lat** – Latitude in decimal degrees North.
- **locality** – A locality within the administrative region, such as a town or city.
- **lon** – Longitude in decimal degrees East.
- **postalcode** – A code used for postal delivery.
- **region** – An administrative region of the nation, such as a state or province.
- **room** – A particular room in a building.
- **speed** – The speed at which the entity is moving, in meters per second.
- **street** – A thoroughfare within the locality, or a crossing of two thoroughfares.
- **text** – A catch-all element that captures any other information about the location.
- **timestamp** – UTC timestamp specifying the moment when the reading was taken.
- **uri** – A URI or URL pointing to information about the location.
- **options** – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0080.stanza' from '/home/docs/checkouts/readthedocs.org/branches/master/slixmpp/plugins/xep_0080/stanza.py'>
```

```
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

 Clear existing user location information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0080.stanza.Geoloc(xml=None, parent=None)
```

XMPP's <geoloc> stanza allows entities to know the current geographical or physical location of an entity.
(XEP-0080: User Location)

Example <geoloc> stanzas:

```
<geoloc xmlns='http://jabber.org/protocol/geoloc'>
```

```
<geoloc xmlns='http://jabber.org/protocol/geoloc' xml:lang='en'>
    <accuracy>20</accuracy>
    <country>Italy</country>
    <lat>45.44</lat>
    <locality>Venice</locality>
    <lon>12.33</lon>
</geoloc>
```

Stanza Interface:

accuracy	--- Horizontal GPS error in meters.
alt	--- Altitude in meters above or below sea level.
area	--- A named area such as a campus or neighborhood.
bearing	--- GPS bearing (direction in which the entity is heading to reach its next waypoint), measured in decimal degrees relative to true north.
building	--- A specific building on a street or in an area.
country	--- The nation where the user is located.
countrycode	--- The ISO 3166 two-letter country code.
datum	--- GPS datum.
description	--- A natural-language name for or description of the location.
error	--- Horizontal GPS error in arc minutes. Obsoleted by the accuracy parameter.
floor	--- A particular floor in a building.
lat	--- Latitude in decimal degrees North.
locality	--- A locality within the administrative region, such as a town or city.
lon	--- Longitude in decimal degrees East.
postalcode	--- A code used for postal delivery.
region	--- An administrative region of the nation, such as a state or province.
room	--- A particular room in a building.
speed	--- The speed at which the entity is moving, in meters per second.
street	--- A thoroughfare within the locality, or a crossing of two thoroughfares.
text	--- A catch-all element that captures any other information about the location.
timestamp	--- UTC timestamp specifying the moment when the reading was taken.
uri	--- A URI or URL pointing to information about the location.

exception(e)

Override exception passback for presence.

get_accuracy()

Return the value of the <accuracy> element as an integer.

get_alt()

Return the value of the <alt> element as an integer.

get_bearing()

Return the value of the <bearing> element as a float.

get_error()

Return the value of the <error> element as a float.

get_lat()

Return the value of the <lat> element as a float.

get_lon()

Return the value of the <lon> element as a float.

get_speed()

Return the value of the <speed> element as a float.

get_timestamp()

Return the value of the <timestampe> element as a DateTime.

interfaces = {'accuracy', 'alt', 'area', 'bearing', 'building', 'country', 'countrycode'}

name = 'geoloc'

namespace = 'http://jabber.org/protocol/geoloc'

plugin_attrib = 'geoloc'

set_accuracy(accuracy)

Set the value of the <accuracy> element.

Parameters **accuracy** – Horizontal GPS error in meters

set_alt(alt)

Set the value of the <alt> element.

Parameters **alt** – Altitude in meters above or below sea level

set_bearing(bearing)

Set the value of the <bearing> element.

Parameters **bearing** – GPS bearing (direction in which the entity is heading to reach its next waypoint), measured in decimal degrees relative to true north

set_error(error)

Set the value of the <error> element.

Parameters **error** – Horizontal GPS error in arc minutes; this element is deprecated in favor of <accuracy/>

set_lat(lat)

Set the value of the <lat> element.

Parameters **lat** – Latitude in decimal degrees North

set_lon(lon)

Set the value of the <lon> element.

Parameters **lon** – Longitude in decimal degrees East

set_speed(*speed*)

Set the value of the <speed> element.

Parameters **speed** – The speed at which the entity is moving, in meters per second

set_timestamp(*timestamp*)

Set the value of the <timestamp> element.

Parameters **timestamp** – UTC timestamp specifying the moment when the reading was taken

```
sub_interfaces = {'accuracy', 'alt', 'area', 'bearing', 'building', 'country', 'country'}
```

XEP 0082

```
class slixmpp.plugins.xep_0082.XEP_0082(xmpp, config=None)
```

XEP-0082: XMPP Date and Time Profiles

XMPP uses a subset of the formats allowed by ISO 8601 as a matter of pragmatism based on the relatively few formats historically used by the XMPP.

Also see <<http://www.xmpp.org/extensions/xep-0082.html>>.

Methods: date – Create a time stamp using the Date profile. datetime – Create a time stamp using the DateTime profile. time – Create a time stamp using the Time profile. format_date – Format an existing date object. format_datetime – Format an existing datetime object. format_time – Format an existing time object. parse – Convert a time string into a Python datetime object.

XEP 0084

```
class slixmpp.plugins.xep_0084.XEP_0084(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0084.stanza' from '/home/docs/checkouts/readthed
```

```
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

Clear existing avatar metadata information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0084.stanza.Data(xml=None, parent=None)
```

```
    del_value()
```

```
    get_value()
```

```
    interfaces = {'value'}
```

```
    name = 'data'
```

```
    namespace = 'urn:xmpp:avatar:data'
```

```
    plugin_attrib = 'avatar_data'
```

```
    set_value(value)
```

```

class slixmpp.plugins.xep_0084.stanza.Info (xml=None, parent=None)
    interfaces = {'bytes', 'height', 'id', 'type', 'url', 'width'}
    name = 'info'
    namespace = 'urn:xmpp:avatar:metadata'
    plugin_attrib = 'info'
    plugin_multi_attrib = 'items'
class slixmpp.plugins.xep_0084.stanza.MetaData (xml=None, parent=None)
    add_info (id, itype, ibytes, height=None, width=None, url=None)
    add_pointer (xml)
    interfaces = {}
    name = 'metadata'
    namespace = 'urn:xmpp:avatar:metadata'
    plugin_attrib = 'avatar_metadata'
    plugin_attrib_map = {'info': <class 'slixmpp.plugins.xep_0084.stanza.Info'>, 'items': None}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0084.stanza.Info'>, <class 'slixmpp.plugins.xep_0084.stanza.MetaData'>}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi_stanza'>}
class slixmpp.plugins.xep_0084.stanza.Pointer (xml=None, parent=None)
    interfaces = {}
    name = 'pointer'
    namespace = 'urn:xmpp:avatar:metadata'
    plugin_attrib = 'pointer'
    plugin_multi_attrib = 'pointers'

```

XEP 0085

```

class slixmpp.plugins.xep_0085.XEP_0085 (xmpp, config=None)
    XEP-0085 Chat State Notifications
    stanza = <module 'slixmpp.plugins.xep_0085.stanza' from '/home/docs/checkouts/readthedocs.org/branch/master/slixmpp/plugins/xep_0085/stanza.py'>

```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0085.stanza.Active(xml=None, parent=None)
```

```
    name = 'active'
```

```
class slixmpp.plugins.xep_0085.stanza.ChatState(xml=None, parent=None)
```

Example chat state stanzas:

```
<message>
    <active xmlns="http://jabber.org/protocol/chatstates" />
</message>

<message>
    <paused xmlns="http://jabber.org/protocol/chatstates" />
</message>
```

```
def chat_state():
    get_chat_state()
    interfaces = {'chat_state'}
    is_extension = True
    name = ''
    namespace = 'http://jabber.org/protocol/chatstates'
    plugin_attrib = 'chat_state'
    set_chat_state(state)
    setup(xml=None)
```

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
states = {'active', 'composing', 'gone', 'inactive', 'paused'}
sub_interfaces = {'chat_state'}
```

```
class slixmpp.plugins.xep_0085.stanza.Composing(xml=None, parent=None)
```

```
    name = 'composing'
```

```
class slixmpp.plugins.xep_0085.stanza.Gone(xml=None, parent=None)
```

```
    name = 'gone'
```

```
class slixmpp.plugins.xep_0085.stanza.Inactive(xml=None, parent=None)
```

```
    name = 'inactive'
```

```
class slixmpp.plugins.xep_0085.stanza.Paused(xml=None, parent=None)

    name = 'paused'
```

XEP 0086

```
class slixmpp.plugins.xep_0086.XEP_0086(xmpp, config=None)
XEP-0086: Error Condition Mappings
```

Older XMPP implementations used code based error messages, similar to HTTP response codes. Since then, error condition elements have been introduced. XEP-0086 provides a mapping between the new condition elements and a combination of error types and the older response codes.

Also see <<http://xmpp.org/extensions/xep-0086.html>>.

Configuration Values:

```
override -- Indicates if applying legacy error codes should
        be done automatically. Defaults to True.
        If False, then inserting legacy error codes can
        be done using:
            iq['error']['legacy']['condition'] = ...
```

```
stanza = <module 'slixmpp.plugins.xep_0086.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0086.stanza.LegacyError(xml=None, parent=None)
```

Older XMPP implementations used code based error messages, similar to HTTP response codes. Since then, error condition elements have been introduced. XEP-0086 provides a mapping between the new condition elements and a combination of error types and the older response codes.

Also see <<http://xmpp.org/extensions/xep-0086.html>>.

Example legacy error stanzas:

```
<error xmlns="jabber:client" code="501" type="cancel">
    <feature-not-implemented
        xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
</error>

<error code="402" type="auth">
    <payment-required
        xmlns="urn:ietf:params:xml:ns:xmpp-stanzas" />
</error>
```

Variables `error_map` – A map of error conditions to error types and code values.

```
error_map = {'bad-request': ('modify', '400'), 'conflict': ('cancel', '409'), 'feature-not-implemented': ('cancel', '401')}
interfaces = {'condition'}
name = 'legacy'
```

```
namespace = 'jabber:client'
overrides = ['set_condition']
plugin_attrib = 'legacy'

set_condition(value)
    Set the error type and code based on the given error condition value.

    Parameters value – The new error condition.

setup(xml)
    Don't create XML for the plugin.
```

XEP 0092

```
class slixmpp.plugins.xep_0092.XEP_0092(xmpp, config=None)
XEP-0092: Software Version

get_version(jid, ifrom=None, timeout=None, callback=None, timeout_callback=None)
    Retrieve the software version of a remote agent.

    Arguments: jid – The JID of the entity to query.

stanza = <module 'slixmpp.plugins.xep_0092.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0092.stanza.Version(xml=None, parent=None)
XMPP allows for an agent to advertise the name and version of the underlying software libraries, as well as the operating system that the agent is running on.
```

Example version stanzas:

```
<iq type="get">
    <query xmlns="jabber:iq:version" />
</iq>

<iq type="result">
    <query xmlns="jabber:iq:version">
        <name>Slixmpp</name>
        <version>1.0</version>
        <os>Linux</os>
    </query>
</iq>
```

Stanza Interface:

name	-- The human readable name of the software.
version	-- The specific version of the software.
os	-- The name of the operating system running the program.

```
interfaces = {'name', 'os', 'version'}
name = 'query'
```

```
namespace = 'jabber:iq:version'
plugin_attrib = 'software_version'
sub_interfaces = {'name', 'os', 'version'}
```

XEP 0106

```
class slixmpp.plugins.xep_0106.XEP_0106(xmpp, config=None)
```

XEP 0107

```
class slixmpp.plugins.xep_0107.XEP_0107(xmpp, config=None)
XEP-0107: User Mood
```

```
publish_mood(value=None, text=None, options=None, ifrom=None, callback=None, timeout=None,
              timeout_callback=None)
Publish the user's current mood.
```

Parameters

- **value** – The name of the mood to publish.
- **text** – Optional natural-language description or reason for the mood.
- **options** – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0107.stanza' from '/home/docs/checkouts/readthedoc
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
Clear existing user mood information to stop notifications.
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0107.stanza.UserMood(xml=None, parent=None)
```

```
del_value()
get_value()

interfaces = {'text', 'value'}

moods = {'afraid', 'amazed', 'amorous', 'angry', 'annoyed', 'anxious', 'aroused', 'ashamed',
         'bored', 'calm', 'caring', 'confused', 'curious', 'disgusted', 'drunk', 'dumbfounded',
         'embarrassed', 'excited', 'grateful', 'guilty', 'helpless', 'hopeful', 'hurt', 'impatient',
         'interested', 'jealous', 'kind', 'loved', 'neutral', 'optimistic', 'proud', 'relaxed',
         'relieved', 'sad', 'surprised', 'tired', 'worried'}
name = 'mood'

namespace = 'http://jabber.org/protocol/mood'
plugin_attrib = 'mood'

set_value(value)

sub_interfaces = {'text'}
```

XEP 0108

```
class slixmpp.plugins.xep_0108.XEP_0108(xmpp, config=None)
XEP-0108: User Activity
```

```
publish_activity(general, specific=None, text=None, options=None, ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

 Publish the user's current activity.

Parameters

- **general** – The required general category of the activity.
- **specific** – Optional specific activity being done as part of the general category.
- **text** – Optional natural-language description or reason for the activity.
- **options** – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0108.stanza' from '/home/docs/checkouts/readthed
```

```
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

 Clear existing user activity information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0108.stanza.UserActivity(xml=None, parent=None)
```

```
del_value()
general = {'doing_chores', 'drinking', 'eating', 'exercising', 'grooming', 'having_apple'}
get_value()
interfaces = {'text', 'value'}
name = 'activity'
namespace = 'http://jabber.org/protocol/activity'
plugin_attrib = 'activity'
set_value(value)
specific = {'at_the_spa', 'brushing_teeth', 'buying_groceries', 'cleaning', 'coding', 'driving'}
sub_interfaces = {'text'}
```

XEP 0115

```
class slixmpp.plugins.xep_0115.XEP_0115(xmpp, config=None)
XEP-0115: Entity Capabilities
```

```
stanza = <module 'slixmpp.plugins.xep_0115.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0115.stanza.Capabilities(xml=None, parent=None)
```

```
interfaces = {'ext', 'hash', 'node', 'ver'}
name = 'c'
namespace = 'http://jabber.org/protocol/caps'
plugin_attrib = 'caps'
```

XEP 0118

```
class slixmpp.plugins.xep_0118.XEP_0118(xmpp, config=None)
XEP-0118: User Tune
```

```
publish_tune(artist=None, length=None, rating=None, source=None, title=None, track=None,
             uri=None, options=None, ifrom=None, callback=None, timeout=None, time-
             out_callback=None)
```

Publish the user's current tune.

Parameters

- **artist** – The artist or performer of the song.
- **length** – The length of the song in seconds.
- **rating** – The user's rating of the song (from 1 to 10)
- **source** – The album name, website, or other source of the song.
- **title** – The title of the song.
- **track** – The song's track number, or other unique identifier.
- **uri** – A URL to more information about the song.
- **options** – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0118.stanza' from '/home/docs/checkouts/readthed
```

```
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

Clear existing user tune information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0118.stanza.UserTune(xml=None, parent=None)

    interfaces = {'artist', 'length', 'rating', 'source', 'title', 'track', 'uri'}
    name = 'tune'
    namespace = 'http://jabber.org/protocol/tune'
    plugin_attrib = 'tune'
    set_length(value)
    set_rating(value)
    sub_interfaces = {'artist', 'length', 'rating', 'source', 'title', 'track', 'uri'}
```

XEP 0122

```
class slixmpp.plugins.xep_0122.XEP_0122(xmpp, config=None)
```

XEP-0122: Data Forms

```
stanza = <module 'slixmpp.plugins.xep_0004.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

```
class slixmpp.plugins.xep_0122.stanza.FormValidation(xml=None, parent=None)
```

Validation values for form fields.

Example:

```
<field var='evt.date' type='text-single' label='Event Date/Time'>
    <validate xmlns='http://jabber.org/protocol/xdata-validate'
              datatype='xs:dateTime' />
    <value>2003-10-06T11:22:00-07:00</value>
</field>
```

Questions:

- Should this look at the datatype value and convert the range values as appropriate?
- Should this stanza provide a pass/fail for a value from the field, or convert field value to datatype?

```
get_basic()
get_open()
get_range()
get_regex()

interfaces = {'basic', 'datatype', 'open', 'range', 'regex'}
name = 'validate'

namespace = 'http://jabber.org/protocol/xdata-validate'
```

```

plugin_attrib = 'validate'
plugin_attrib_map = {}
plugin_tag_map = {}
remove_all(except_tag=None)
set_basic(value)
set_open(value)
set_range(value, minimum=None, maximum=None)
set_regex(regex)
sub_interfaces = {'basic', 'open', 'range', 'regex'}

```

XEP 0128

class `slixmpp.plugins.xep_0128.XEP_0128(xmpp, config=None)`
XEP-0128: Service Discovery Extensions

Allow the use of data forms to add additional identity information to disco#info results.

Also see <<http://www.xmpp.org/extensions/xep-0128.html>>.

Variables

- **disco** – A reference to the XEP-0030 plugin.
- **static** – Object containing the default set of static node handlers.

add_extended_info(*jid*=None, *node*=None, ***kwargs*)
Add additional, extended identity information to a node.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **data** – Either a form, or a list of forms to add as extended information.

del_extended_info(*jid*=None, *node*=None, ***kwargs*)
Remove all extended identity information to a node.

Parameters

- **jid** (Optional[*JID*]) – The JID to modify.
- **node** (Optional[str]) – The node to modify.

set_extended_info(*jid*=None, *node*=None, ***kwargs*)
Set additional, extended identity information to a node.

Replaces any existing extended information.

Parameters

- **jid** – The JID to modify.
- **node** – The node to modify.
- **data** – Either a form, or a list of forms to use as extended information, replacing any existing extensions.

XEP 0131

```
class slixmpp.plugins.xep_0131.XEP_0131(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0131.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0131.stanza.Headers(xml=None, parent=None)
```

```
del_headers()
get_headers()
interfaces = {'headers'}
is_extension = True
name = 'headers'
namespace = 'http://jabber.org/protocol/shim'
plugin_attrib = 'headers'
set_headers(values)
```

XEP 0133

```
class slixmpp.plugins.xep_0133.XEP_0133(xmpp, config=None)
```

XEP 0152

```
class slixmpp.plugins.xep_0152.XEP_0152(xmpp, config=None)
```

XEP-0152: Reachability Addresses

```
publish_reachability(addresses, options=None, ifrom=None, callback=None, timeout=None,
                      timeout_callback=None)
```

Publish alternative addresses where the user can be reached.

Parameters

- **addresses** (List[Dict[str, str]]) – A list of dictionaries containing the URI and optional description for each address.
- **options** (Optional[*Form*]) – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0152.stanza' from '/home/docs/checkouts/readthed
```

```
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
```

Clear existing user activity information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2013 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0152.stanza.Address (xml=None, parent=None)

    interfaces = {'desc', 'uri'}
    lang_interfaces = {'desc'}
    name = 'addr'
    namespace = 'urn:xmpp:reach:0'
    plugin_attrib = 'address'
    plugin_multi_attrib = 'addresses'
    sub_interfaces = {'desc'}

class slixmpp.plugins.xep_0152.stanza.Reachability (xml=None, parent=None)

    interfaces = {}
    name = 'reach'
    namespace = 'urn:xmpp:reach:0'
    plugin_attrib = 'reach'
    plugin_attrib_map = {'address': <class 'slixmpp.plugins.xep_0152.stanza.Address'>, 'a'}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0152.stanza.Address'>}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi'>}
```

XEP 0153

```
class slixmpp.plugins.xep_0153.XEP_0153 (xmpp, config=None)

    stanza = <module 'slixmpp.plugins.xep_0153.stanza' from '/home/docs/checkouts/readthedocs.org/branch/develop/lib/python2.7/site-packages/slixmpp/plugins/xep_0153/stanza.py'>
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0153.stanza.VCardTempUpdate (xml=None, parent=None)

    get_photo()
    interfaces = {'photo'}
    name = 'x'
```

```
namespace = 'vcard-temp:x:update'
plugin_attrib = 'vcard_temp_update'
set_photo (value)
sub_interfaces = {'photo'}
```

XEP 0163

```
class slixmpp.plugins.xep_0163.XEP_0163(xmpp, config=None)
XEP-0163: Personal Eventing Protocol (PEP)
```

add_interest (*namespace, jid=None*)

Mark an interest in a PEP subscription by including a disco feature with the ‘+notify’ extension.

Parameters

- **namespace** (*str*) – The base namespace to register as an interest, such as ‘<http://jabber.org/protocol/tune>’. This may also be a list of such namespaces.
- **jid** (*Optional[[JID](#)]*) – Optionally specify the JID.

```
publish(stanza, node=None, id=None, options=None, ifrom=None, timeout_callback=None, call-
back=None, timeout=None)
Publish a PEP update.
```

This is just a (very) thin wrapper around the XEP-0060 publish() method to set the defaults expected by PEP.

Parameters

- **stanza** (*ElementBase*) – The PEP update stanza to publish.
- **node** (*Optional[str]*) – The node to publish the item to. If not specified, the stanza’s namespace will be used.
- **id** (*Optional[str]*) – Optionally specify the ID of the item.
- **options** (*Optional[[Form](#)]*) – A form of publish options.

register_pep (*name, stanza*)

Setup and configure events and stanza registration for the given PEP stanza:

- Add disco feature for the PEP content.
- Register disco interest in the PEP content.
- Map events from the PEP content’s namespace to the given name.

Parameters

- **name** (*str*) – The event name prefix to use for PEP events.
- **stanza** – The stanza class for the PEP content.

remove_interest (*namespace, jid=None*)

Mark an interest in a PEP subscription by including a disco feature with the ‘+notify’ extension.

Parameters

- **namespace** (*str*) – The base namespace to remove as an interest, such as ‘<http://jabber.org/protocol/tune>’. This may also be a list of such namespaces.
- **jid** (*Optional[[JID](#)]*) – Optionally specify the JID.

XEP 0172

```
class slixmpp.plugins.xep_0172.XEP_0172(xmpp, config=None)
XEP-0172: User Nickname
```

```
publish_nick(nick=None, options=None, ifrom=None, timeout_callback=None, callback=None,
            timeout=None)
```

Publish the user's current nick.

Parameters

- **nick** (Optional[str]) – The user nickname to publish.
- **options** (Optional[Form]) – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0172.stanza' from '/home/docs/checkouts/readthed
```

```
stop(ifrom=None, timeout_callback=None, callback=None, timeout=None)
```

Clear existing user nick information to stop notifications.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0172.stanza.UserNick(xml=None, parent=None)
```

XEP-0172: User Nickname allows the addition of a <nick> element in several stanza types, including <message> and <presence> stanzas.

The nickname contained in a <nick> should be the global, friendly or informal name chosen by the owner of a bare JID. The <nick> element may be included when establishing communications with new entities, such as normal XMPP users or MUC services.

The nickname contained in a <nick> element will not necessarily be the same as the nickname used in a MUC.

Example stanzas:

```
<message to="user@example.com">
  <nick xmlns="http://jabber.org/nick/nick">The User</nick>
  <body>...</body>
</message>

<presence to="otheruser@example.com" type="subscribe">
  <nick xmlns="http://jabber.org/nick/nick">The User</nick>
</presence>
```

Stanza Interface:

```
nick -- A global, friendly or informal name chosen by a user.
```

```
del_nick()
```

Remove the <nick> element.

```
get_nick()
```

Return the nickname in the <nick> element.

```
interfaces = {'nick'}
```

```
name = 'nick'
```

```
namespace = 'http://jabber.org/protocol/nick'
```

```
plugin_attrib = 'nick'  
set_nick(nick)  
    Add a <nick> element with the given nickname.  
  
Arguments: nick – A human readable, informal name.
```

XEP 0184

```
class slixmpp.plugins.xep_0184.XEP_0184(xmpp, config=None)  
XEP-0184: Message Delivery Receipts
```

```
ack(msg)  
    Acknowledge a message by sending a receipt.  
  
Arguments: msg – The message to acknowledge.
```

```
stanza = <module 'slixmpp.plugins.xep_0184.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Erik Reuterborg Larsson, Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0184.stanza.Received(xml=None, parent=None)
```

```
del_receipt()  
get_receipt()  
interfaces = {'receipt'}  
is_extension = True  
name = 'received'  
namespace = 'urn:xmpp:receipts'  
plugin_attrib = 'receipt'  
set_receipt(value)
```

setup(xml=None)
Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
sub_interfaces = {'receipt'}  
class slixmpp.plugins.xep_0184.stanza.Request(xml=None, parent=None)  
  
del_request_receipt()  
get_request_receipt()  
interfaces = {'request_receipt'}
```

```
is_extension = True
name = 'request'
namespace = 'urn:xmpp:receipts'
plugin_attrib = 'request_receipt'
set_request_receipt(val)
setup(xml=None)
    Initialize the stanza's XML contents.
    Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating new XML.

sub_interfaces = {'request_receipt'}
```

XEP 0186

```
class slixmpp.plugins.xep_0186.XEP_0186(xmpp, config=None)
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp. See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0186.stanza.Invisible(xml=None, parent=None)

interfaces = {}
name = 'invisible'
namespace = 'urn:xmpp:invisible:0'
plugin_attrib = 'invisible'

class slixmpp.plugins.xep_0186.stanza.Visible(xml=None, parent=None)

interfaces = {}
name = 'visible'
namespace = 'urn:xmpp:visible:0'
plugin_attrib = 'visible'
```

XEP 0191

```
class slixmpp.plugins.xep_0191.XEP_0191(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0191.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0191.stanza.Block(xml=None, parent=None)
```

```
name = 'block'  
plugin_attrib = 'block'
```

```
class slixmpp.plugins.xep_0191.stanza.BlockList(xml=None, parent=None)
```

```
del_items()  
get_items()  
interfaces = {'items'}  
name = 'blocklist'  
namespace = 'urn:xmpp:blocking'  
plugin_attrib = 'blocklist'  
set_items(values)
```

```
class slixmpp.plugins.xep_0191.stanza.Unblock(xml=None, parent=None)
```

```
name = 'unblock'  
plugin_attrib = 'unblock'
```

XEP 0196

```
class slixmpp.plugins.xep_0196.XEP_0196(xmpp, config=None)
```

XEP-0196: User Gaming

```
publish_gaming(name=None, level=None, server_name=None, uri=None, character_name=None,  
character_profile=None, server_address=None, options=None, ifrom=None, call-  
back=None, timeout=None, timeout_callback=None)
```

Publish the user's current gaming status.

Parameters

- **name** (Optional[str]) – The name of the game.
- **level** (Optional[str]) – The user's level in the game.
- **uri** (Optional[str]) – A URI for the game or relevant gaming service
- **server_name** (Optional[str]) – The name of the server where the user is playing.

- **server_address** (Optional[str]) – The hostname or IP address of the server where the user is playing.
- **character_name** (Optional[str]) – The name of the user's character in the game.
- **character_profile** (Optional[str]) – A URI for a profile of the user's character.
- **options** (Optional[Form]) – Optional form of publish options.

```
stanza = <module 'slixmpp.plugins.xep_0196.stanza' from '/home/docs/checkouts/readthedocs.org/branches/master/slixmpp/plugins/xep_0196/stanza.py'>
stop(ifrom=None, callback=None, timeout=None, timeout_callback=None)
Clear existing user gaming information to stop notifications.
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0196.stanza.UserGaming(xml=None, parent=None)
```

```
interfaces = {'character_name', 'character_profile', 'level', 'name', 'server_address'}
name = 'game'
namespace = 'urn:xmpp:gaming:0'
plugin_attrib = 'gaming'
sub_interfaces = {'character_name', 'character_profile', 'level', 'name', 'server_address'}
```

XEP 0198

```
class slixmpp.plugins.xep_0198.XEP_0198(xmpp, config=None)
XEP-0198: Stream Management
```

```
disconnected(event)
Reset enabled state until we can resume/reenable.
```

```
request_ack(e=None)
Request an ack from the server.
```

```
send_ack()
Send the current ack count to the server.
```

```
session_end(event)
Reset stream management state.
```

```
stanza = <module 'slixmpp.plugins.xep_0198.stanza' from '/home/docs/checkouts/readthedocs.org/branches/master/slixmpp/plugins/xep_0198/stanza.py'>
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0198.stanza.Ack(stream=None,      xml=None,      stype=None,
                                             sto=None,     sfrom=None,    sid=None,     parent=None)
```

```
get_h()
```

```
interfaces = {'h'}
```

```
name = 'a'
```

```
namespace = 'urn:xmpp:sm:3'
```

```
set_h(val)
```

```
setup(xml)
```

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0198.stanza.Enable(stream=None,      xml=None,      stype=None,
                                              sto=None,     sfrom=None,    sid=None,     parent=None)
```

```
get_resume()
```

```
interfaces = {'max', 'resume'}
```

```
name = 'enable'
```

```
namespace = 'urn:xmpp:sm:3'
```

```
set_resume(val)
```

```
setup(xml)
```

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0198.stanza.Enabled(stream=None,      xml=None,      stype=None,
                                                sto=None,     sfrom=None,    sid=None,     parent=None)
```

```
get_resume()
```

```
interfaces = {'id', 'location', 'max', 'resume'}
```

```
name = 'enabled'
```

```
namespace = 'urn:xmpp:sm:3'
```

```
set_resume(val)
```

```

setup (xml)
    Initialize the stanza's XML contents.

    Will return True if XML was generated according to the stanza's definition instead of building a stanza
    object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
    new XML.

class slixmpp.plugins.xep_0198.stanza.Failed (stream=None, xml=None, stype=None,
                                              sto=None, sfrom=None, sid=None, par-
                                              ent=None)

interfaces = {}
name = 'failed'
namespace = 'urn:xmpp:sm:3'

setup (xml)
    Initialize the stanza's XML contents.

    Will return True if XML was generated according to the stanza's definition instead of building a stanza
    object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
    new XML.

class slixmpp.plugins.xep_0198.stanza.RequestAck (stream=None, xml=None,
                                              stype=None, sto=None, sfrom=None,
                                              sid=None, parent=None)

interfaces = {}
name = 'r'
namespace = 'urn:xmpp:sm:3'

setup (xml)
    Initialize the stanza's XML contents.

    Will return True if XML was generated according to the stanza's definition instead of building a stanza
    object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
    new XML.

class slixmpp.plugins.xep_0198.stanza.Resume (stream=None, xml=None, stype=None,
                                              sto=None, sfrom=None, sid=None, par-
                                              ent=None)

get_h ()
interfaces = {'h', 'previd'}
name = 'resume'
namespace = 'urn:xmpp:sm:3'

set_h (val)
setup (xml)
    Initialize the stanza's XML contents.

```

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0198.stanza.Resumed(stream=None, xml=None, stype=None,
                                               sto=None, sfrom=None, sid=None, parent=None)
```

`get_h()`

`interfaces = {'h', 'previd'}`

`name = 'resumed'`

`namespace = 'urn:xmpp:sm:3'`

`set_h(val)`

`setup(xml)`

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0198.stanza.StreamManagement(xml=None, parent=None)
```

`del_optional()`

`del_required()`

`get_optional()`

`get_required()`

`interfaces = {'optional', 'required'}`

`name = 'sm'`

`namespace = 'urn:xmpp:sm:3'`

`plugin_attrib = 'sm'`

`set_optional(val)`

`set_required(val)`

XEP 0199

```
class slixmpp.plugins.xep_0199.XEP_0199(xmpp, config=None)
```

XEP-0199: XMPP Ping

Given that XMPP is based on TCP connections, it is possible for the underlying connection to be terminated without the application's awareness. Ping stanzas provide an alternative to whitespace based keepalive methods for detecting lost connections.

Also see <<http://www.xmpp.org/extensions/xep-0199.html>>.

Attributes:

keepalive – If True, periodically send ping requests to the server. If a ping is not answered, the connection will be reset.

interval – Time in seconds between keepalive pings. Defaults to 300 seconds.

timeout – Time in seconds to wait for a ping response. Defaults to 30 seconds.

Methods:

send_ping – Send a ping to a given JID, returning the round trip time.

async ping (*jid=None, ifrom=None, timeout=None*)

Send a ping request and calculate RTT. This is a coroutine.

Parameters **jid** (Optional[*JID*]) – The JID that will receive the ping.

Return type **float**

send_ping (*jid, ifrom=None, timeout=None, callback=None, timeout_callback=None*)

Send a ping request.

Parameters **jid** (*JID*) – The JID that will receive the ping.

stanza = <module 'slixmpp.plugins.xep_0199.stanza' from '/home/docs/checkouts/readthed

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

class *slixmpp.plugins.xep_0199.stanza.Ping* (*xml=None, parent=None*)

Given that XMPP is based on TCP connections, it is possible for the underlying connection to be terminated without the application's awareness. Ping stanzas provide an alternative to whitespace based keepalive methods for detecting lost connections.

Example ping stanza:

```
<iq type="get">
  <ping xmlns="urn:xmpp:ping" />
</iq>
```

```
interfaces = {}
name = 'ping'
namespace = 'urn:xmpp:ping'
plugin_attrib = 'ping'
```

XEP 0202

class *slixmpp.plugins.xep_0202.XEP_0202* (*xmpp, config=None*)

XEP-0202: Entity Time

get_entity_time (*to, ifrom=None, **iqargs*)

Request the time from another entity.

Parameters **to** (*JID*) – JID of the entity to query.

stanza = <module 'slixmpp.plugins.xep_0202.stanza' from '/home/docs/checkouts/readthed

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2010 Nathanael C. Fritz This file is part of Slixmpp.

See the file LICENSE for copying permission.

class slixmpp.plugins.xep_0202.stanza.EntityTime (*xml=None, parent=None*)

The <time> element represents the local time for an XMPP agent. The time is expressed in UTC to make synchronization easier between entities, but the offset for the local timezone is also included.

Example <time> stanzas:

```
<iq type="result">
  <time xmlns="urn:xmpp:time">
    <utc>2011-07-03T11:37:12.234569</utc>
    <tzo>-07:00</tzo>
  </time>
</iq>
```

Stanza Interface:

```
time -- The local time for the entity (updates utc and tzo).
utc -- The UTC equivalent to local time.
tzo -- The local timezone offset from UTC.
```

del_time()

Remove both the UTC and TZO fields.

get_time()

Return the entity's local time based on the UTC and TZO data.

get_tzo()

Return the timezone offset from UTC as a tzinfo object.

get_utc()

Return the time in UTC as a datetime object.

interfaces = {'time', 'tzo', 'utc'}

name = 'time'

namespace = 'urn:xmpp:time'

plugin_attrib = 'entity_time'

set_time(*value*)

Set both the UTC and TZO fields given a time object.

Parameters **value** – A datetime object or properly formatted string equivalent.

set_tzo(*value*)

Set the timezone offset from UTC.

Parameters **value** – Either a tzinfo object or the number of seconds (positive or negative) to offset.

set_utc(*value*)

Set the time in UTC.

Parameters **value** – A datetime object or properly formatted string equivalent.

sub_interfaces = {'time', 'tzo', 'utc'}

XEP 0203

```
class slixmpp.plugins.xep_0203.XEP_0203(xmpp, config=None)
XEP-0203: Delayed Delivery
```

XMPP stanzas are sometimes withheld for delivery due to the recipient being offline, or are resent in order to establish recent history as is the case with MUCS. In any case, it is important to know when the stanza was originally sent, not just when it was last received.

Also see <<http://www.xmpp.org/extensions/xep-0203.html>>.

```
stanza = <module 'slixmpp.plugins.xep_0203.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0203.stanza.Delay(xml=None, parent=None)
```

```
def_text()
get_from()
get_stamp()
get_text()

interfaces = {'from', 'stamp', 'text'}
name = 'delay'
namespace = 'urn:xmpp:delay'
plugin_attrib = 'delay'
set_from(value)
set_stamp(value)
set_text(value)
```

XEP 0221

```
class slixmpp.plugins.xep_0221.XEP_0221(xmpp, config=None)
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0221.stanza.Media(xml=None, parent=None)
```

```
add_uri(value, itype)
interfaces = {'alt', 'height', 'width'}
name = 'media'
```

```
namespace = 'urn:xmpp:media-element'
plugin_attrib = 'media'
plugin_attrib_map = {'uri': <class 'slixmpp.plugins.xep_0221.stanza.URI'>, 'uris': <class 'slixmpp.plugins.xep_0221.stanza.URIS'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0221.stanza.URI'>}
plugin_overrides = {}
plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi_stanza_base'>}

class slixmpp.plugins.xep_0221.stanza.URI(xml=None, parent=None)

    def __init__(self, xml=None, parent=None):
        super().__init__(xml, parent)
        self._value = None
        self._type = None
        self._name = 'uri'

    @property
    def value(self):
        return self._value

    @value.setter
    def value(self, value):
        self._value = value
        self._type = 'uri'

    @property
    def type(self):
        return self._type

    @type.setter
    def type(self, value):
        self._type = value
        self._name = value

    @property
    def name(self):
        return self._name
```

XEP 0222

```
class slixmpp.plugins.xep_0222.XEP_0222(xmpp, config=None)
```

XEP-0222: Persistent Storage of Public Data via PubSub

```
configure(node, ifrom=None, callback=None, timeout=None)
```

Update a node's configuration to match the public storage profile.

```
retrieve(node, id=None, item_ids=None, ifrom=None, callback=None, timeout=None)
```

Retrieve public data via PEP.

This is just a (very) thin wrapper around the XEP-0060 publish() method to set the defaults expected by PEP.

Parameters

- **node** (str) – The node to retrieve content from.
- **id** (Optional[str]) – Optionally specify the ID of the item.
- **item_ids** (Optional[List[str]]) – Specify a group of IDs. If id is also specified, it will be included in item_ids.
- **ifrom** (Optional[[JID](#)]) – Specify the sender's JID.
- **timeout** (Optional[int]) – The length of time (in seconds) to wait for a response before exiting the send call if blocking is used. Defaults to `slixmpp.xmlstream.RESPONSE_TIMEOUT`
- **callback** (Optional[Callable]) – Optional reference to a stream handler function. Will be executed when a reply stanza is received.

store (*stanza, node=None, id=None, ifrom=None, options=None, callback=None, timeout=None*)
Store public data via PEP.

This is just a (very) thin wrapper around the XEP-0060 publish() method to set the defaults expected by PEP.

Parameters

- **stanza** (*ElementBase*) – The private content to store.
- **node** (Optional[str]) – The node to publish the content to. If not specified, the stanza's namespace will be used.
- **id** (Optional[str]) – Optionally specify the ID of the item.
- **options** (Optional[*Form*]) – Publish options to use, which will be modified to fit the persistent storage option profile.

XEP 0223

class slixmpp.plugins.xep_0223.XEP_0223 (*xmpp, config=None*)

XEP-0223: Persistent Storage of Private Data via PubSub

configure (*node, ifrom=None, callback=None, timeout=None*)

Update a node's configuration to match the public storage profile.

retrieve (*node, id=None, item_ids=None, ifrom=None, callback=None, timeout=None, time-out_callback=None*)

Retrieve private data via PEP.

This is just a (very) thin wrapper around the XEP-0060 publish() method to set the defaults expected by PEP.

Parameters

- **node** (str) – The node to retrieve content from.
- **id** (Optional[str]) – Optionally specify the ID of the item.
- **item_ids** (Optional[List[str]]) – Specify a group of IDs. If id is also specified, it will be included in item_ids.
- **ifrom** (Optional[JID]) – Specify the sender's JID.
- **timeout** (Optional[int]) – The length of time (in seconds) to wait for a response before exiting the send call if blocking is used. Defaults to slixmpp.xmlstream.RESPONSE_TIMEOUT
- **callback** (Optional[Callable]) – Optional reference to a stream handler function. Will be executed when a reply stanza is received.

store (*stanza, node=None, id=None, ifrom=None, options=None, callback=None, timeout=None, time-out_callback=None*)

Store private data via PEP.

This is just a (very) thin wrapper around the XEP-0060 publish() method to set the defaults expected by PEP.

Parameters

- **stanza** (*ElementBase*) – The private content to store.
- **node** (Optional[str]) – The node to publish the content to. If not specified, the stanza's namespace will be used.

- **id** (Optional[str]) – Optionally specify the ID of the item.
- **options** (Optional[*Form*]) – Publish options to use, which will be modified to fit the persistent storage option profile.

XEP 0224

```
class slixmpp.plugins.xep_0224.XEP_0224(xmpp, config=None)
```

XEP-0224: Attention

```
request_attention(to, mfrom=None, mbody="")
```

Send an attention message with an optional body.

Arguments: to – The attention request recipient's JID. mfrom – Optionally specify the sender of the attention request. mbody – An optional message body to include in the request.

```
stanza = <module 'slixmpp.plugins.xep_0224.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0224.stanza.Attention(xml=None, parent=None)
```

```
del_attention()
get_attention()
interfaces = {'attention'}
is_extension = True
name = 'attention'
namespace = 'urn:xmpp:attention:0'
plugin_attrib = 'attention'
set_attention(value)
```

```
setup(xml)
```

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters **xml** – An existing XML object to use for the stanza's content instead of generating new XML.

XEP 0231

```
class slixmpp.plugins.xep_0231.XEP_0231(xmpp, config=None)
    XEP-0231 Bits of Binary
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz Emmanuel Gil Peyrot <linkmauve@linkmauve.fr> This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0231.stanza.BitsOfBinary(xml=None, parent=None)
```

```
def __init__(self, xmpp):
    self.xmpp = xmpp
    self._data = {}
    self._max_age = None
    self._type = None

    self._interfaces = {'cid', 'data', 'max_age', 'type'}
    self._name = 'data'
    self._namespace = 'urn:xmpp:bits-of-binary'
    self._plugin_attrib = 'bits-of-binary'

    self._set_data(self._data)

    self._set_max_age(self._max_age)

    self._set_type(self._type)

    self._set_interfaces(self._interfaces)

    self._set_name(self._name)

    self._set_namespace(self._namespace)

    self._set_plugin_attrib(self._plugin_attrib)
```

XEP 0235

```
class slixmpp.plugins.xep_0235.XEP_0235(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0235.stanza' from '/home/docs/checkouts/readthedocstheme/_build/html/_static/stanza.py'>
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0235.stanza.OAuth(xml=None, parent=None)
```

```
def __init__(self, xmpp):
    self.xmpp = xmpp
    self._signature = None
    self._method = 'HMAC-SHA1'

    self._interfaces = {'oauth_consumer_key', 'oauth_nonce', 'oauth_signature', 'oauth_signature_method'}
    self._name = 'oauth'
    self._namespace = 'urn:xmpp:oauth:0'
    self._plugin_attrib = 'oauth'

    self._generate_signature(self._signature, self._method)

    self._set_interfaces(self._interfaces)

    self._set_name(self._name)

    self._set_namespace(self._namespace)

    self._set_plugin_attrib(self._plugin_attrib)

    self._set_signature(self._signature)

    self._set_method(self._method)

    self._set_interfaces(self._interfaces)

    self._set_name(self._name)

    self._set_namespace(self._namespace)

    self._set_plugin_attrib(self._plugin_attrib)

    self._set_signature(self._signature)

    self._set_method(self._method)
```

XEP 0249

```
class slixmpp.plugins.xep_0249.XEP_0249(xmpp, config=None)
```

XEP-0249: Direct MUC Invitations

```
send_invitation(jid, roomjid, password=None, reason=None, *, mfrom=None)
```

Send a direct MUC invitation to an XMPP entity.

Parameters

- **jid** ([JID](#)) – The JID of the entity that will receive the invitation
- **roomjid** ([JID](#)) – the address of the groupchat room to be joined
- **password** (*str*) – a password needed for entry into a password-protected room (OPTIONAL).
- **reason** (*str*) – a human-readable purpose for the invitation (OPTIONAL).

```
stanza = <module 'slixmpp.plugins.xep_0249.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2011 Nathanael C. Fritz, Dalek This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0249.stanza.Invite(xml=None, parent=None)
```

XMPP allows for an agent in an MUC room to directly invite another user to join the chat room (as opposed to a mediated invitation done through the server).

Example invite stanza:

```
<message from='crone1@shakespeare.lit/desktop'
    to='hecate@shakespeare.lit'>
    <x xmlns='jabber:x:conference'
        jid='darkcave@macbeth.shakespeare.lit'
        password='cauldronburn'
        reason='Hey Hecate, this is the place for all good witches!' />
</message>
```

Stanza Interface:

```
jid      -- The JID of the groupchat room
password -- The password used to gain entry in the room
           (optional)
reason   -- The reason for the invitation (optional)
```

```
interfaces = ('jid', 'password', 'reason')
name = 'x'
namespace = 'jabber:x:conference'
plugin_attrib = 'groupchat_invite'
```

XEP 0256

```
class slixmpp.plugins.xep_0256.XEP_0256(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0012.stanza' from '/home/docs/checkouts/readthed
```

XEP 0257

```
class slixmpp.plugins.xep_0257.XEP_0257(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0257.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0257.stanza.AppendCert(xml=None, parent=None)
```

```
del_cert_management()
get_cert_management()
interfaces = {'cert_management', 'name', 'x509cert'}
name = 'append'
namespace = 'urn:xmpp:saslcert:1'
plugin_attrib = 'sasl_cert_append'
set_cert_management(value)
sub_interfaces = {'name', 'x509cert'}
```

```
class slixmpp.plugins.xep_0257.stanza.CertItem(xml=None, parent=None)
```

```
del_users()
get_users()
interfaces = {'name', 'users', 'x509cert'}
name = 'item'
namespace = 'urn:xmpp:saslcert:1'
plugin_attrib = 'item'
plugin_multi_attrib = 'items'
set_users(values)
sub_interfaces = {'name', 'x509cert'}
```

```
class slixmpp.plugins.xep_0257.stanza.Certs(xml=None, parent=None)
```

```
interfaces = {}
```

```
name = 'items'
namespace = 'urn:xmpp:saslcert:1'
plugin_attrib = 'sasl_certs'
plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0257.stanza.CertItem'>, 'item': <class 'slixmpp.plugins.xep_0257.stanza.CertItem'>}
plugin_iterables = {<class 'slixmpp.plugins.xep_0257.stanza.CertItem'>}
plugin_overrides = {}
plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi_stanza_base'>}

class slixmpp.plugins.xep_0257.stanza.DisableCert(xml=None, parent=None)

    interfaces = {'name'}
    name = 'disable'
    namespace = 'urn:xmpp:saslcert:1'
    plugin_attrib = 'sasl_cert_disable'
    sub_interfaces = {'name'}

class slixmpp.plugins.xep_0257.stanza.RevokeCert(xml=None, parent=None)

    interfaces = {'name'}
    name = 'revoke'
    namespace = 'urn:xmpp:saslcert:1'
    plugin_attrib = 'sasl_cert_revoke'
    sub_interfaces = {'name'}
```

XEP 0258

```
class slixmpp.plugins.xep_0258.XEP_0258(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0258.stanza' from '/home/docs/checkouts/readthedocs.org/branch/develop/lib/python2.7/site-packages/slixmpp/plugins/xep_0258/stanza.py'>
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0258.stanza.Catalog(xml=None, parent=None)
```

```
get_from()
get_restrict()
get_to()
interfaces = {'desc', 'from', 'id', 'name', 'restrict', 'size', 'to'}
name = 'catalog'
```

```

namespace = 'urn:xmpp:sec-label:catalog:2'
plugin_attrib = 'security_label_catalog'
plugin_attrib_map = {'item': <class 'slixmpp.plugins.xep_0258.stanza.CatalogItem'>, ...}
plugin_iterables = {<class 'slixmpp.plugins.xep_0258.stanza.CatalogItem'>}
plugin_overrides = {}
plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi...'}
set_from(value)
set_restrict(value)
set_to(value)

class slixmpp.plugins.xep_0258.stanza.CatalogItem(xml=None, parent=None)

    get_default()
    interfaces = {'default', 'selector'}
    name = 'catalog'
    namespace = 'urn:xmpp:sec-label:catalog:2'
    plugin_attrib = 'item'
    plugin_attrib_map = {'security_label': <class 'slixmpp.plugins.xep_0258.stanza.Securi...'}
    plugin_iterables = {}
    plugin_multi_attrib = 'items'
    plugin_overrides = {}
    plugin_tag_map = {'{urn:xmpp:sec-label:0}securitylabel': <class 'slixmpp.plugins.xep_0258....'}
    set_default(value)

class slixmpp.plugins.xep_0258.stanza.DisplayMarking(xml=None, parent=None)

    del_value()
    get_bgcolor()
    get_fgcolor()
    get_value()
    interfaces = {'bgcolor', 'fgcolor', 'value'}
    name = 'displaymarking'
    namespace = 'urn:xmpp:sec-label:0'
    plugin_attrib = 'display_marking'
    set_value(value)

class slixmpp.plugins.xep_0258.stanza.ESSLLabel(xml=None, parent=None)

    del_value()
    get_value()

```

```
interfaces = {'value'}
name = 'esssecuritylabel'
namespace = 'urn:xmpp:sec-label:ess:0'
plugin_attrib = 'ess'
set_value(value)

class slixmpp.plugins.xep_0258.stanza.EquivalentLabel(xml=None, parent=None)

    name = 'equivalentlabel'
    namespace = 'urn:xmpp:sec-label:0'
    plugin_attrib = 'equivalent_label'
    plugin_attrib_map = {'ess': <class 'slixmpp.plugins.xep_0258.stanza.ESSLabel'>}
    plugin_iterables = {}
    plugin_multi_attrib = 'equivalent_labels'
    plugin_overrides = {}
    plugin_tag_map = {'{urn:xmpp:sec-label:ess:0}esssecuritylabel': <class 'slixmpp.plugins.xep_0258.stanza.EssSecurityLabel'>}

class slixmpp.plugins.xep_0258.stanza.Label(xml=None, parent=None)

    name = 'label'
    namespace = 'urn:xmpp:sec-label:0'
    plugin_attrib = 'label'
    plugin_attrib_map = {'ess': <class 'slixmpp.plugins.xep_0258.stanza.ESSLabel'>}
    plugin_iterables = {}
    plugin_overrides = {}
    plugin_tag_map = {'{urn:xmpp:sec-label:ess:0}esssecuritylabel': <class 'slixmpp.plugins.xep_0258.stanza.EssSecurityLabel'>}

class slixmpp.plugins.xep_0258.stanza.SecurityLabel(xml=None, parent=None)

    add_equivalent(label)
    name = 'securitylabel'
    namespace = 'urn:xmpp:sec-label:0'
    plugin_attrib = 'security_label'
    plugin_attrib_map = {'display_marking': <class 'slixmpp.plugins.xep_0258.stanza.DisplayMarking'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0258.stanza.EquivalentLabel'>}
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi_stanza_base'>}
```

XEP 0279

```
class slixmpp.plugins.xep_0279.XEP_0279(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0279.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0279.stanza.IPCheck(xml=None, parent=None)
```

```
del_ip_check()
get_ip_check()
interfaces = {'ip_check'}
is_extension = True
name = 'ip'
namespace = 'urn:xmpp:sic:0'
plugin_attrib = 'ip_check'
set_ip_check(value)
```

XEP 0280

```
class slixmpp.plugins.xep_0280.XEP_0280(xmpp, config=None)
```

XEP-0280 Message Carbons

```
stanza = <module 'slixmpp.plugins.xep_0280.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0280.stanza.CarbonDisable(xml=None, parent=None)
```

```
interfaces = {}
name = 'disable'
namespace = 'urn:xmpp:carbons:2'
plugin_attrib = 'carbon_disable'
```

```
class slixmpp.plugins.xep_0280.stanza.CarbonEnable(xml=None, parent=None)
```

```
interfaces = {}
name = 'enable'
```

```
namespace = 'urn:xmpp:carbons:2'
plugin_attrib = 'carbon_enable'

class slixmpp.plugins.xep_0280.stanza.PrivateCarbon(xml=None, parent=None)

    interfaces = {}
    name = 'private'
    namespace = 'urn:xmpp:carbons:2'
    plugin_attrib = 'carbon_private'

class slixmpp.plugins.xep_0280.stanza.ReceivedCarbon(xml=None, parent=None)

    del_carbon_received()
    get_carbon_received()
    interfaces = {'carbon_received'}
    is_extension = True
    name = 'received'
    namespace = 'urn:xmpp:carbons:2'
    plugin_attrib = 'carbon_received'
    set_carbon_received(stanza)

class slixmpp.plugins.xep_0280.stanza.SentCarbon(xml=None, parent=None)

    del_carbon_sent()
    get_carbon_sent()
    interfaces = {'carbon_sent'}
    is_extension = True
    name = 'sent'
    namespace = 'urn:xmpp:carbons:2'
    plugin_attrib = 'carbon_sent'
    set_carbon_sent(stanza)
```

XEP 0297

```
class slixmpp.plugins.xep_0297.XEP_0297(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0297.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0297.stanza.Forwarded(xml=None, parent=None)

    del_stanza()
    get_stanza()
    interfaces = {'stanza'}
    name = 'forwarded'
    namespace = 'urn:xmpp:forward:0'
    plugin_attrib = 'forwarded'
    set_stanza(value)
```

XEP 0300

```
class slixmpp.plugins.xep_0300.XEP_0300(*args, **kwargs)
```

```
stanza = <module 'slixmpp.plugins.xep_0300.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2017 Emmanuel Gil Peyrot This file is part of Slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0300.stanza.Hash(xml=None, parent=None)

    allowed_algos = ['sha-1', 'sha-256', 'sha-512', 'sha3-256', 'sha3-512', 'BLAKE2b256',
        del_value()
        get_value()
        interfaces = {'algo', 'value'}
        name = 'hash'
        namespace = 'urn:xmpp:hashes:2'
        plugin_attrib = 'hash'
        set_algo(value)
        set_value(value)
```

XEP 0308

```
class slixmpp.plugins.xep_0308.XEP_0308(xmpp, config=None)
XEP-0308 Last Message Correction
```

```
stanza = <module 'slixmpp.plugins.xep_0308.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0308.stanza.Replace(xml=None, parent=None)
```

```
interfaces = {'id'}
name = 'replace'
namespace = 'urn:xmpp:message-correct:0'
plugin_attrib = 'replace'
```

XEP 0313

```
class slixmpp.plugins.xep_0313.XEP_0313(xmpp, config=None)
XEP-0313 Message Archive Management
```

```
retrieve(jid=None, start=None, end=None, with_jid=None, ifrom=None, reverse=False, timeout=None, callback=None, iterator=False, rsm=None)
Send a MAM query and retrieve the results.
```

Parameters

- **jid** ([JID](#)) – Entity holding the MAM records
- **start**, **end** ([datetime](#)) – MAM query temporal boundaries
- **with_jid** ([JID](#)) – Filter results on this JID
- **ifrom** ([JID](#)) – To change the from address of the query
- **reverse** ([bool](#)) – Get the results in reverse order
- **timeout** ([int](#)) – IQ timeout
- **callback** ([func](#)) – Custom callback for handling results
- **iterator** ([bool](#)) – Use RSM and iterate over a paginated query
- **rsm** ([dict](#)) – RSM custom options

Return type

Awaitable

```
stanza = <module 'slixmpp.plugins.xep_0313.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.
See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0313.stanza.Fin (xml=None, parent=None)
    name = 'fin'
    namespace = 'urn:xmpp:mam:2'
    plugin_attrib = 'mam_fin'

class slixmpp.plugins.xep_0313.stanza.MAM (xml=None, parent=None)
    del_results()
    get_end()
    get_results()
    get_start()
    get_with()
    interfaces = {'end', 'queryid', 'results', 'start', 'with'}
    name = 'query'
    namespace = 'urn:xmpp:mam:2'
    plugin_attrib = 'mam'
    set_end(value)
    set_results(values)
    set_start(value)
    set_with(value)
    setup(xml=None)
        Initialize the stanza's XML contents.
    Will return True if XML was generated according to the stanza's definition instead of building a stanza
    object from an existing XML object.

    Parameters xml – An existing XML object to use for the stanza's content instead of generating
    new XML.

    sub_interfaces = {'end', 'start', 'with'}
```

class slixmpp.plugins.xep_0313.stanza.Preferences (*xml=None, parent=None*)

```
    get_always()
    get_never()
    interfaces = {'always', 'default', 'never'}
    name = 'prefs'
    namespace = 'urn:xmpp:mam:2'
    plugin_attrib = 'mam_prefs'
```

```
    set_always (value)
    set_never (value)
    sub_interfaces = {'always', 'never'}
class slixmpp.plugins.xep_0313.stanza.Result (xml=None, parent=None)

    interfaces = {'id', 'queryid'}
    name = 'result'
    namespace = 'urn:xmpp:mam:2'
    plugin_attrib = 'mam_result'
```

XEP 0319

```
class slixmpp.plugins.xep_0319.XEP_0319 (xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0319.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2013 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0319.stanza.Idle (xml=None, parent=None)

    get_since()
    interfaces = {'since'}
    name = 'idle'
    namespace = 'urn:xmpp:idle:1'
    plugin_attrib = 'idle'
    set_since (value)
```

XEP 0332

```
class slixmpp.plugins.xep_0332.XEP_0332 (xmpp, config=None)
```

XEP-0332: HTTP over XMPP transport

```
default_config = {'supported_headers': ['Accept', 'Accept-Charset', 'Accept-Encoding']}
TODO: Do we really need to mention the supported_headers?!
```

```
dependencies = {'xep_0030', 'xep_0131'}
xep_0047 not included. xep_0001, 0137 and 0166 are missing
```

Stanza elements

Slixmpp: The Slick XMPP Library Implementation of HTTP over XMPP transport <http://xmpp.org/extensions/xep-0332.html> Copyright (C) 2015 Riptide IO, sangeeth@riptideio.com This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0332.stanza.data.HTTPData (xml=None, parent=None)  
    The data element.
```

```
    get_data (encoding='text')  
  
    interfaces = {'data'}  
  
    is_extension = True  
  
    name = 'data'  
  
    namespace = 'urn:xmpp:http'  
  
    plugin_attrib = 'data'  
  
    set_data (data, encoding='text')
```

slixmpp: The Slick XMPP Library Implementation of HTTP over XMPP transport <http://xmpp.org/extensions/xep-0332.html> Copyright (C) 2015 Riptide IO, sangeeth@riptideio.com This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0332.stanza.request.HTTPRequest (xml=None, parent=None)
```

All HTTP communication is done using the *Request/Response* paradigm. Each HTTP Request is made sending an *iq* stanza containing a *req* element to the server. Each *iq* stanza sent is of type *set*.

Examples:

```
<iq type='set' from='a@b.com/browser' to='x@y.com' id='1'>  
  <req xmlns='urn:xmpp:http'  
        method='GET'  
        resource='/api/users'  
        version='1.1'>  
    <headers xmlns='http://jabber.org/protocol/shim'>  
        <header name='Host'>b.com</header>  
    </headers>  
  </req>  
</iq>  
  
<iq type='set' from='a@b.com/browser' to='x@y.com' id='2'>  
  <req xmlns='urn:xmpp:http'  
        method='PUT'  
        resource='/api/users'  
        version='1.1'>  
    <headers xmlns='http://jabber.org/protocol/shim'>  
        <header name='Host'>b.com</header>  
        <header name='Content-Type'>text/html</header>  
        <header name='Content-Length'>...</header>  
    </headers>  
    <data>  
        <text>...</text>  
    </data>  
  </req>  
</iq>
```

```
get_method()
get_resource()
get_version()

interfaces = {'method', 'resource', 'version'}
name = 'request'

namespace = 'urn:xmpp:http'
plugin_attrib = 'http-req'

set_method(method)
set_resource(resource)
set_version(version='1.1')
```

Slixmpp: The Slick XMPP Library Implementation of HTTP over XMPP transport <http://xmpp.org/extensions/xep-0332.html> Copyright (C) 2015 Riptide IO, sangeeth@riptideio.com This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0332.stanza.response.HTTPResponse(xml=None, parent=None)
```

When the HTTP Server responds, it does so by sending an *iq* stanza response (type='result') back to the client containing the *resp* element. Since responses are asynchronous, and since multiple requests may be active at the same time, responses may be returned in a different order than the in which the original requests were made.

Examples:

```
<iq type='result'
    from='httpserver@clayster.com'
    to='httpclient@clayster.com/browser' id='2'>
    <resp xmlns='urn:xmpp:http'
        version='1.1'
        statusCode='200'
        statusMessage='OK'>
        <headers xmlns='http://jabber.org/protocol/shim'>
            <header name='Date'>Fri, 03 May 2013 16:39:54 GMT-4</header>
            <header name='Server'>Clayster</header>
            <header name='Content-Type'>text/turtle</header>
            <header name='Content-Length'>...</header>
            <header name='Connection'>Close</header>
        </headers>
        <data>
            <text>
                ...
            </text>
        </data>
    </resp>
</iq>
```

```
get_code()
get_message()

interfaces = {'code', 'message', 'version'}
name = 'response'

namespace = 'urn:xmpp:http'
```

```
plugin_attrib = 'http-resp'
set_code(code)
set_message(message)
set_version(version='1.1')
```

XEP 0333

```
class slixmpp.plugins.xep_0333.XEP_0333(xmpp, config=None)
```

```
send_marker(mto, id, marker, thread=None, *, mfrom=None)
```

Send a chat marker.

Parameters

- **mto** ([JID](#)) – recipient of the marker
- **id** (*str*) – Identifier of the marked message
- **marker** (*str*) – Marker to send (one of displayed, retrieved, or acknowledged)
- **thread** (*str*) – Message thread
- **mfrom** (*str*) – Use a specific JID to send the message

```
stanza = <module 'slixmpp.plugins.xep_0333.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

slixmpp: The Slick XMPP Library Copyright (C) 2016 Emmanuel Gil Peyrot This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0333.stanza.Acknowledged(xml=None, parent=None)
```

```
interfaces = {'id'}
name = 'acknowledged'
namespace = 'urn:xmpp:chat-markers:0'
plugin_attrib = 'acknowledged'

class slixmpp.plugins.xep_0333.stanza.Displayed(xml=None, parent=None)
```

```
interfaces = {'id'}
name = 'displayed'
namespace = 'urn:xmpp:chat-markers:0'
plugin_attrib = 'displayed'
```

```
class slixmpp.plugins.xep_0333.stanza.Markable(xml=None, parent=None)
```

```
name = 'markable'
namespace = 'urn:xmpp:chat-markers:0'
```

```
    plugin_attrib = 'markable'

class slixmpp.plugins.xep_0333.stanza.Received(xml=None, parent=None)

    interfaces = {'id'}
    name = 'received'
    namespace = 'urn:xmpp:chat-markers:0'
    plugin_attrib = 'received'
```

XEP 0334

```
class slixmpp.plugins.xep_0334.XEP_0334(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0334.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

slixmpp: The Slick XMPP Library Implementation of Message Processing Hints <http://xmpp.org/extensions/xep-0334.html> This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0334.stanza.NoCopy(xml=None, parent=None)
```

```
    name = 'no-copy'
    namespace = 'urn:xmpp:hints'
    plugin_attrib = 'no-copy'
```

```
class slixmpp.plugins.xep_0334.stanza.NoPermanentStore(xml=None, parent=None)
```

```
    name = 'no-permanent-store'
    namespace = 'urn:xmpp:hints'
    plugin_attrib = 'no-permanent-store'
```

```
class slixmpp.plugins.xep_0334.stanza.NoStore(xml=None, parent=None)
```

```
    name = 'no-store'
    namespace = 'urn:xmpp:hints'
    plugin_attrib = 'no-store'
```

```
class slixmpp.plugins.xep_0334.stanza.Store(xml=None, parent=None)
```

```
    name = 'store'
    namespace = 'urn:xmpp:hints'
    plugin_attrib = 'store'
```

XEP 0335

```
class slixmpp.plugins.xep_0335.XEP_0335(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0335.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2018 Maxime “pep” Buquet This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0335.stanza.JSON_Container(xml=None, parent=None)
```

```
    del_value()
    get_value()
    interfaces = {'value'}
    name = 'json'
    namespace = 'urn:xmpp:json:0'
    plugin_attrib = 'json'
    set_value(value)
```

XEP 0352

```
class slixmpp.plugins.xep_0352.XEP_0352(xmpp, config=None)
```

XEP-0352: Client State Indication

```
    send_active()
        Send an ‘active’ state
    send_inactive()
        Send an ‘active’ state
```

```
stanza = <module 'slixmpp.plugins.xep_0352.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2012 Nathanael C. Fritz, Lance J.T. Stout This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0352.stanza.Active(stream=None, xml=None, stype=None,
                                              sto=None, sfrom=None, sid=None, parent=None)

    name = 'active'
    namespace = 'urn:xmpp:csi:0'
    plugin_attrib = 'active'
```

setup (xml)

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

```
class slixmpp.plugins.xep_0352.stanza.ClientStateIndication (xml=None, parent=None)
```

```
    name = 'csi'
```

```
    namespace = 'urn:xmpp:csi:0'
```

```
    plugin_attrib = 'csi'
```

```
class slixmpp.plugins.xep_0352.stanza.Inactive (stream=None, xml=None, stype=None, sto=None, sfrom=None, sid=None, parent=None)
```

```
    name = 'inactive'
```

```
    namespace = 'urn:xmpp:csi:0'
```

```
    plugin_attrib = 'inactive'
```

setup (xml)

Initialize the stanza's XML contents.

Will return True if XML was generated according to the stanza's definition instead of building a stanza object from an existing XML object.

Parameters `xml` – An existing XML object to use for the stanza's content instead of generating new XML.

XEP 0353

```
class slixmpp.plugins.xep_0353.XEP_0353 (xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0353.stanza' from '/home/docs/checkouts/readthedoc
```

Stanza elements

slixmpp: The Slick XMPP Library Copyright (C) 2020 Emmanuel Gil Peyrot This file is part of slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0353.stanza.Accept (xml=None, parent=None)
```

```
    name = 'accept'
```

```
    plugin_attrib = 'jingle_accept'
```

```
class slixmpp.plugins.xep_0353.stanza.JingleMessage (xml=None, parent=None)
```

```
    interfaces = {'id'}
```

```

namespace = 'urn:xmpp:jingle-message:0'

class slixmpp.plugins.xep_0353.stanza.Proceed(xml=None, parent=None)

    name = 'proceed'
    plugin_attrib = 'jingle_proceed'

class slixmpp.plugins.xep_0353.stanza.Propose(xml=None, parent=None)

    del_descriptions()
    get_descriptions()

        Return type List[Tuple[str, str]]

    interfaces = {'descriptions', 'id'}
    name = 'propose'
    plugin_attrib = 'jingle_propose'
    set_descriptions(descriptions)

class slixmpp.plugins.xep_0353.stanza.Reject(xml=None, parent=None)

    name = 'reject'
    plugin_attrib = 'jingle_reject'

class slixmpp.plugins.xep_0353.stanza.Retract(xml=None, parent=None)

    name = 'retract'
    plugin_attrib = 'jingle_retract'

```

XEP 0359

```
class slixmpp.plugins.xep_0359.XEP_0359(xmpp, config=None)
```

XEP-0359: Unique and Stable Stanza IDs

```
stanza = <module 'slixmpp.plugins.xep_0359.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0359.stanza.OriginID(xml=None, parent=None)
```

```

interfaces = {'id'}
name = 'origin-id'
namespace = 'urn:xmpp:sid:0'
plugin_attrib = 'origin_id'
```

```
class slixmpp.plugins.xep_0359.stanza.StanzaID (xml=None, parent=None)

    interfaces = {'by', 'id'}
    name = 'stanza-id'
    namespace = 'urn:xmpp:sid:0'
    plugin_attrib = 'stanza_id'

slixmpp.plugins.xep_0359.stanza.register_plugins()
```

XEP 0363

Stanza elements

slixmpp: The Slick XMPP Library Copyright (C) 2018 Emmanuel Gil Peyrot This file is part of slixmpp.
See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0363.stanza.Get (xml=None, parent=None)

    interfaces = {'url'}
    name = 'get'
    namespace = 'urn:xmpp:http:upload:0'
    plugin_attrib = 'get'

class slixmpp.plugins.xep_0363.stanza.Header (xml=None, parent=None)

    del_value()
    get_value()

    interfaces = {'name', 'value'}
    name = 'header'
    namespace = 'urn:xmpp:http:upload:0'
    plugin_attrib = 'header'
    plugin_multi_attrib = 'headers'
    set_value (value)

class slixmpp.plugins.xep_0363.stanza.Put (xml=None, parent=None)

    interfaces = {'url'}
    name = 'put'
    namespace = 'urn:xmpp:http:upload:0'
    plugin_attrib = 'put'

class slixmpp.plugins.xep_0363.stanza.Request (xml=None, parent=None)

    interfaces = {'content-type', 'filename', 'size'}
```

```

name = 'request'
namespace = 'urn:xmpp:http:upload:0'
plugin_attrib = 'http_upload_request'

class slixmpp.plugins.xep_0363.stanza.Slot(xml=None, parent=None)

    name = 'slot'
    namespace = 'urn:xmpp:http:upload:0'
    plugin_attrib = 'http_upload_slot'

```

XEP 0369

class slixmpp.plugins.xep_0369.XEP_0369(xmpp, config=None)
XEP-0369: MIX-CORE

async can_create_channel(service)

Check if the current user can create a channel on the MIX service

Parameters `service` (`JID`) – MIX service jid

Return type bool

async create_channel(service, channel=None, *, ifrom=None, **iqkwargs)

Create a MIX channel.

Parameters

- `service` (`JID`) – MIX service JID
- `channel` (*Optional[str]*) – Channel name (or leave empty to let the service generate it)

Return type str

Returns The channel name, as created by the service

async destroy_channel(channel, *, ifrom=None, **iqkwargs)

Destroy a MIX channel. :param JID channel: MIX channelJID

async get_channel_info(channel)

” Get the contents of the channel info node.

Parameters `channel` (`JID`) – The MIX channel

Return type Dict[str, Any]

Returns a dict containing the last modified time and form contents (Name, Description, Contact per the spec, YMMV)

async join_channel(channel, nick, subscribe=None, *, ifrom=None, **iqkwargs)

Join a MIX channel.

Parameters

- `channel` (`JID`) – JID of the MIX channel
- `nick` (`str`) – Desired nickname on that channel
- `subscribe` (`Set[str]`) – Set of nodes to subscribe to when joining. If empty, all nodes will be subscribed by default.

Return type `Set[str]`

Returns The nodes that failed to subscribe, if any

```
async leave_channel(channel, *, ifrom=None, **iqkwargs)
    "Leave a MIX channel :param JID channel: JID of the channel to leave
```

Return type `None`

```
async list_channels(service, *, ifrom=None, **discokwargs)
    List the channels on a MIX service
```

Parameters `service` (`JID`) – MIX service JID

Return type `List[Tuple[JID, str]]`

Returns A list of channels with their JID and name

```
async list_mix_nodes(channel, ifrom=None, **discokwargs)
    List mix nodes for a channel.
```

Parameters `channel` (`JID`) – The MIX channel

Return type `Set[str]`

Returns List of nodes available

```
async list_participants(channel, *, ifrom=None, **pubsubkwargs)
    List the participants of a MIX channel :param JID channel: The MIX channel
```

Return type `List[Tuple[str, str, Optional[JID]]]`

Returns A list of tuples containing the participant id, nick, and jid (if available)

```
async set_nick(channel, nick, *, ifrom=None, **iqkwargs)
```

Set your nick on a channel. The returned nick MAY be different from the one provided, depending on service configuration. :param JID channel: MIX channel JID :param str nick: desired nick :rtype: str :return: The nick saved on the MIX channel

```
stanza = <module 'slixmpp.plugins.xep_0369.stanza' from '/home/docs/checkouts/readthedocs.org/user_builds/slixmpp/env/lib/python3.6/site-packages/slixmpp/plugins/xep_0369/stanza.py'>
```

```
async update_subscription(channel, subscribe=None, unsubscribe=None, *, ifrom=None,
                           **iqkwargs)
```

Update a MIX channel subscription.

Parameters

- `channel` (`JID`) – JID of the MIX channel
- `subscribe` (`Set[str]`) – Set of notes to subscribe to additionally.
- `unsubscribe` (`Set[str]`) – Set of notes to unsubscribe from.

Return type `Tuple[Set[str], Set[str]]`

Returns A tuple containing the set of nodes that failed to subscribe and the set of nodes that failed to unsubscribe.

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0369.stanza.Create(xml=None, parent=None)

    interfaces = {'channel'}
    name = 'create'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_create'

class slixmpp.plugins.xep_0369.stanza.Destroy(xml=None, parent=None)

    interfaces = {'channel'}
    name = 'destroy'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_destroy'

class slixmpp.plugins.xep_0369.stanza.Join(xml=None, parent=None)

    interfaces = {'id', 'nick'}
    name = 'join'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_join'
    sub_interfaces = {'nick'}

class slixmpp.plugins.xep_0369.stanza.Leave(xml=None, parent=None)

    name = 'leave'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_leave'

class slixmpp.plugins.xep_0369.stanza.MIX(xml=None, parent=None)

    interfaces = {'jid', 'nick'}
    name = 'mix'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix'
    sub_interfaces = {'jid', 'nick'}

class slixmpp.plugins.xep_0369.stanza.Participant(xml=None, parent=None)

    interfaces = {'jid', 'nick'}
```

```
name = 'participant'
namespace = 'urn:xmpp:mix:core:1'
plugin_attrib = 'mix_participant'
sub_interfaces = {'jid', 'nick'}

class slixmpp.plugins.xep_0369.stanza.Setnick (xml=None, parent=None)

    interfaces = {'nick'}
    name = 'setnick'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_setnick'
    sub_interfaces = {'nick'}

class slixmpp.plugins.xep_0369.stanza.Subscribe (xml=None, parent=None)

    interfaces = {'node'}
    name = 'subscribe'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'subscribe'

class slixmpp.plugins.xep_0369.stanza.Unsubscribe (xml=None, parent=None)

    interfaces = {'node'}
    name = 'unsubscribe'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'unsubscribe'

class slixmpp.plugins.xep_0369.stanza.UpdateSubscription (xml=None, parent=None)

    interfaces = {'jid'}
    name = 'update-subscription'
    namespace = 'urn:xmpp:mix:core:1'
    plugin_attrib = 'mix_updatesub'

slixmpp.plugins.xep_0369.stanza.register_plugins()
```

XEP 0377

```
class slixmpp.plugins.xep_0377.XEP_0377(xmpp, config=None)
XEP-0377: Spam reporting
```

```
stanza = <module 'slixmpp.plugins.xep_0377.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0377.stanza.Report(xml=None, parent=None)
A spam/abuse report.
```

Example sub stanza:

```
<report xmlns="urn:xmpp:reporting:0">
    <text xml:lang="en">
        Never came trouble to my house like this.
    </text>
    <spam/>
</report>
```

Stanza Interface:

abuse	— Flag the report as abuse
spam	— Flag the report as spam
text	— Add a reason to the report

Only one <spam/> or <abuse/> element can be present at once.

```
get_abuse()
get_spam()
interfaces = ('spam', 'abuse', 'text')
name = 'report'
namespace = 'urn:xmpp:reporting:0'
plugin_attrib = 'report'
set_abuse(value)
set_spam(value)
sub_interfaces = {'text'}
class slixmpp.plugins.xep_0377.stanza.Text(xml=None, parent=None)

name = 'text'
namespace = 'urn:xmpp:reporting:0'
plugin_attrib = 'text'
```

XEP 0380

```
class slixmpp.plugins.xep_0380.XEP_0380(xmpp, config=None)
XEP-0380: Explicit Message Encryption
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2016 Emmanuel Gil Peyrot <linkmauve@linkmauve.fr> This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0380.stanza.Encryption(xml=None, parent=None)
```

```
interfaces = {'name', 'namespace'}
name = 'encryption'
namespace = 'urn:xmpp:eme:0'
plugin_attrib = 'eme'
```

XEP 0394

```
class slixmpp.plugins.xep_0394.XEP_0394(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0394.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2017 Emmanuel Gil Peyrot <linkmauve@linkmauve.fr> This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0394.stanza.BlockCode(xml=None, parent=None)
```

```
name = 'bcode'
plugin_attrib = 'bcode'
plugin_multi_attrib = 'bcodes'
```

```
class slixmpp.plugins.xep_0394.stanza.BlockQuote(xml=None, parent=None)
```

```
name = 'bquote'
plugin_attrib = 'bquote'
plugin_multi_attrib = 'bquotes'
```

```
class slixmpp.plugins.xep_0394.stanza.CodeType(xml=None, parent=None)
```

```
name = 'code'
plugin_attrib = 'code'
```

```

class slixmpp.plugins.xep_0394.stanza.DeletedType (xml=None, parent=None)

    name = 'deleted'
    plugin_attrib = 'deleted'

class slixmpp.plugins.xep_0394.stanza.EmphasisType (xml=None, parent=None)

    name = 'emphasis'
    plugin_attrib = 'emphasis'

class slixmpp.plugins.xep_0394.stanza.Li (xml=None, parent=None)

    get_start()
    interfaces = {'start'}
    name = 'li'
    namespace = 'urn:xmpp:markup:0'
    plugin_attrib = 'li'
    plugin_multi_attrib = 'lis'
    set_start(value)

class slixmpp.plugins.xep_0394.stanza.List (xml=None, parent=None)

    interfaces = {'end', 'li', 'start'}
    name = 'list'
    plugin_attrib = 'list'
    plugin_attrib_map = {'li': <class 'slixmpp.plugins.xep_0394.stanza.Li'>, 'lis': <class 'slixmpp.plugins.xep_0394.stanza.Lis'>}
    plugin_iterables = {<class 'slixmpp.plugins.xep_0394.stanza.Li'>}
    plugin_multi_attrib = 'lists'
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi'>}

class slixmpp.plugins.xep_0394.stanza.Markup (xml=None, parent=None)

    name = 'markup'
    namespace = 'urn:xmpp:markup:0'
    plugin_attrib = 'markup'
    plugin_attrib_map = {'bcode': <class 'slixmpp.plugins.xep_0394.stanza.BlockCode'>, 'b': <class 'slixmpp.plugins.xep_0394.stanza.Block'>,
    plugin_iterables = {<class 'slixmpp.plugins.xep_0394.stanza.BlockQuote'>, <class 'slixmpp.plugins.xep_0394.stanza.BlockText'>,
    plugin_overrides = {}
    plugin_tag_map = {'{jabber:client}stanza': <class 'slixmpp.xmlstream.stanzabase.multi'>}

class slixmpp.plugins.xep_0394.stanza.Span (xml=None, parent=None)

```

```
det_types()
get_types()
interfaces = {'end', 'start', 'types'}
name = 'span'
plugin_attrib = 'span'
plugin_attrib_map = {'code':  <class 'slixmpp.plugins.xep_0394.stanza.CodeType'>, 'dele
plugin_iterables = {}
plugin_multi_attrib = 'spans'
plugin_overrides = {}
plugin_tag_map = {'{urn:xmpp:markup:0}code':  <class 'slixmpp.plugins.xep_0394.stanza.
set_types(value)
```

XEP 0403

```
class slixmpp.plugins.xep_0403.XEP_0403(xmpp, config=None)
XEP-0403: MIX-Presence

stanza = <module 'slixmpp.plugins.xep_0403.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0403.stanza.MIXPresence(xml=None, parent=None)

interfaces = {'jid', 'nick'}
name = 'mix'
namespace = 'urn:xmpp:mix:presence:0'
plugin_attrib = 'mix'
sub_interfaces = {'jid', 'nick'}
slixmpp.plugins.xep_0403.stanza.register_plugins()
```

XEP 0404

```
class slixmpp.plugins.xep_0404.XEP_0404(xmpp, config=None)
XEP-0404: MIX JID Hidden Channels
```

```
async get_anon_by_id(channel, *, ifrom=None, **pubsubkwargs)
Get the jid-participant mapping, by participant id
```

Parameters `channel (JID)` – MIX channel JID

Return type `Dict[str, JID]`

```
async get_anon_by_jid(channel, *, ifrom=None, **pubsubkwargs)
    Get the jid-participant mapping, by JID
```

Parameters `channel (JID)` – MIX channel JID

Return type `Dict[JID, str]`

```
async get_anon_raw(channel, *, ifrom=None, **pubsubkwargs)
    Get the jid-participant mapping result (raw). :param JID channel: MIX channel JID
```

Return type `Iq`

```
async get_preferences(channel, *, ifrom=None, **iqkwargs)
    Get channel preferences with default values. :param JID channel: MIX channel JID
```

Return type `Form`

```
async set_preferences(channel, form, *, ifrom=None, **iqkwargs)
    Set channel preferences :param JID channel: MIX channel JID :param Form form: A 0004 form with
```

updated preferences

Return type `Form`

```
stanza = <module 'slixmpp.plugins.xep_0404.stanza' from '/home/docs/checkouts/readthedoc
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0404.stanza.Participant(xml=None, parent=None)
```

```
interfaces = {'jid'}
name = 'participant'
namespace = 'urn:xmpp:mix:anon:0'
plugin_attrib = 'anon_participant'
sub_interfaces = {'jid'}
```

```
class slixmpp.plugins.xep_0404.stanza.UserPreference(xml=None, parent=None)
```

```
name = 'user-preference'
namespace = 'urn:xmpp:mix:anon:0'
plugin_attrib = 'user_preference'
```

```
slixmpp.plugins.xep_0404.stanza.register_plugins()
```

XEP 0405

```
class slixmpp.plugins.xep_0405.XEP_0405(xmpp, config=None)
XEP-0405: MIX-PAM
```

```
async check_server_capability()
Check if the server is MIX-PAM capable
```

Return type bool

```
async join_channel(room, nick, subscribe=None, *, ito=None, ifrom=None, **iqkwargs)
Join a MIX channel.
```

Parameters

- **room** ([JID](#)) – JID of the MIX channel
- **nick** ([str](#)) – Desired nickname on that channel
- **subscribe** ([Set \[str\]](#)) – Set of nodes to subscribe to when joining. If empty, all nodes will be subscribed by default.

Return type [Set\[str\]](#)

Returns The nodes that failed to subscribe, if any

```
async leave_channel(room, *, ito=None, ifrom=None, **iqkwargs)
Leave a MIX channel :param JID room: JID of the channel to leave
```

Return type Iq

```
stanza = <module 'slixmpp.plugins.xep_0405.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0405.stanza.ClientJoin(xml=None, parent=None)
```

```
interfaces = {'channel'}
name = 'client-join'
namespace = 'urn:xmpp:mix:pam:2'
plugin_attrib = 'client_join'
```

```
class slixmpp.plugins.xep_0405.stanza.ClientLeave(xml=None, parent=None)
```

```
interfaces = {'channel'}
name = 'client-leave'
namespace = 'urn:xmpp:mix:pam:2'
plugin_attrib = 'client_leave'
```

```
slixmpp.plugins.xep_0405.stanza.register_plugins()
```

XEP 0421

```
class slixmpp.plugins.xep_0421.XEP_0421(xmpp, config=None)
```

XEP-0421: Anonymous unique occupant identifiers for MUCs

```
stanza = <module 'slixmpp.plugins.xep_0421.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 “Maxime “pep” Buquet <pep@bouah.net>” This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0421.stanza.OccupantId(xml=None, parent=None)
```

An Occupant-id tag.

An <occupant-id> tag is set by the MUC.

This is useful in semi-anon MUCs (and MUC-PMs) as a stable identifier to prevent the usual races with nicknames.

Without occupant-id, getting the following messages from MUC history would prevent a client from asserting senders are the same entity:

```
<message type='groupchat' from='foo@muc/nick1' id='message1'>
    <body>Some message</body>
    <occupant-id xmlns='urn:xmpp:occupant-id:0' id='unique-opaque-id1' />
</message>
<message type='groupchat' from='foo@muc/nick2' id='message2'>
    <body>Some correction</body>
    <occupant-id xmlns='urn:xmpp:occupant-id:0' id='unique-opaque-id1' />
    <replace xmlns='urn:xmpp:message-correct:0' id='message1' />
</message>
```

```
interface = {'id'}
name = 'occupant-id'
namespace = 'urn:xmpp:occupant-id:0'
plugin_attrib = 'occupant-id'
```

XEP 0422

```
class slixmpp.plugins.xep_0422.XEP_0422(xmpp, config=None)
```

XEP-0422: Message Fastening

```
stanza = <module 'slixmpp.plugins.xep_0422.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0422.stanza.ApplyTo(xml=None, parent=None)

    interfaces = {'id', 'shell'}
    name = 'apply-to'
    namespace = 'urn:xmpp:fasten:0'
    plugin_attrib = 'apply_to'
    set_shell(value)

class slixmpp.plugins.xep_0422.stanza.External(xml=None, parent=None)

    interfaces = {'name'}
    name = 'external'
    namespace = 'urn:xmpp:fasten:0'
    plugin_attrib = 'external'

slixmpp.plugins.xep_0422.stanza.register_plugins()
```

XEP 0424

```
class slixmpp.plugins.xep_0424.XEP_0424(xmpp, config=None)
```

XEP-0424: Message Retraction

```
send_retraction(mto, id, mtype='chat', include_fallback=True, fallback_text=None, *,  
                mfrom=None)
```

Send a message retraction

Parameters

- **mto** ([JID](#)) – The JID to retract the message from
- **id** (*str*) – Message ID to retract
- **mtype** (*str*) – Message type
- **include_fallback** (*bool*) – Whether to include a fallback body
- **fallback_text** (*Optional[str]*) – The content of the fallback body. None will set the default value.

```
stanza = <module 'slixmpp.plugins.xep_0424.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0424.stanza.Retract(xml=None, parent=None)

    name = 'retract'
    namespace = 'urn:xmpp:message-retract:0'
    plugin_attrib = 'retract'

class slixmpp.plugins.xep_0424.stanza.Retracted(xml=None, parent=None)

    interfaces = {'stamp'}
    name = 'retracted'
    namespace = 'urn:xmpp:message-retract:0'
    plugin_attrib = 'retracted'

slixmpp.plugins.xep_0424.stanza.register_plugins()
```

XEP 0425

```
class slixmpp.plugins.xep_0425.XEP_0425(xmpp, config=None)
    XEP-0425: Message Moderation

    stanza = <module 'slixmpp.plugins.xep_0425.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0425.stanza.Moderate(xml=None, parent=None)

    interfaces = {'reason'}
    name = 'moderate'
    namespace = 'urn:xmpp:message-moderate:0'
    plugin_attrib = 'moderate'
    sub_interfaces = {'reason'}

class slixmpp.plugins.xep_0425.stanza.Moderated(xml=None, parent=None)

    interfaces = {'by', 'reason'}
    name = 'moderated'
```

```
namespace = 'urn:xmpp:message-moderate:0'
plugin_attrib = 'moderated'
sub_interfaces = {'reason'}
slixmpp.plugins.xep_0425.stanza.register_plugins()
```

XEP 0428

```
class slixmpp.plugins.xep_0428.XEP_0428(xmpp, config=None)
    XEP-0428: Fallback Indication
    stanza = <module 'slixmpp.plugins.xep_0428.stanza' from '/home/docs/checkouts/readthed...
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0428.stanza.Fallback(xml=None, parent=None)

    name = 'fallback'
    namespace = 'urn:xmpp:fallback:0'
    plugin_attrib = 'fallback'
slixmpp.plugins.xep_0428.stanza.register_plugins()
```

XEP 0437

```
class slixmpp.plugins.xep_0437.XEP_0437(xmpp, config=None)
```

```
stanza = <module 'slixmpp.plugins.xep_0437.stanza' from '/home/docs/checkouts/readthed...
subscribe(service, *, pfrom=None)
    Subscribe to room activity on a MUC service. :param JID service: MUC service
unsubscribe(service, *, pfrom=None)
    Unsubscribe from room activity on a MUC service. :param JID service: MUC service
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0437.stanza.Activity(xml=None, parent=None)

    name = 'activity'
    namespace = 'urn:xmpp:rai:0'
    plugin_attrib = 'activity'
```

```
class slixmpp.plugins.xep_0437.stanza.RAI (xml=None, parent=None)
    def_activities()
    get_activities()
        Return type Iterable[JID]
    interfaces = {'activities'}
    name = 'rai'
    namespace = 'urn:xmpp:rai:0'
    plugin_attrib = 'rai'
    set_activities (activities)
    slixmpp.plugins.xep_0437.stanza.register_plugins()
```

XEP 0439

```
class slixmpp.plugins.xep_0439.XEP_0439 (xmpp, config=None)
    XEP-0439: Quick Response
```

ask_for_actions (*mto, body, actions, mtype='chat', lang=None, *, mfrom=None*)
Send a message with a set of actions.

Parameters

- **mto** (JID) – The JID of the entity which will receive the message
- **body** (str) – The message body of the question
- **str]] actions** (Iterable[Tuple[str,]) – A set of tuples containing (action, label) for each action
- **mtype** (str) – The message type
- **lang** (str) – The lang of the message (if not use, the default for this session will be used).

ask_for_response (*mto, body, responses, mtype='chat', lang=None, *, mfrom=None*)
Send a message with a set of responses.

Parameters

- **mto** (JID) – The JID of the entity which will receive the message
- **body** (str) – The message body of the question
- **str]] responses** (Iterable[Tuple[str,]) – A set of tuples containing (value, label) for each response
- **mtype** (str) – The message type
- **lang** (str) – The lang of the message (if not use, the default for this session will be used).

```
stanza = <module 'slixmpp.plugins.xep_0439.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet <mathieui@mathieui.net> This file is part of Slixmpp.

See the file LICENSE for copying permission

```
class slixmpp.plugins.xep_0439.stanza.Action(xml=None, parent=None)

    interfaces = {'id', 'label'}
    name = 'action'
    namespace = 'urn:xmpp:tmp:quick-response'
    plugin_attrib = 'action'

class slixmpp.plugins.xep_0439.stanza.ActionSelected(xml=None, parent=None)

    interfaces = {'id'}
    name = 'action-selected'
    namespace = 'urn:xmpp:tmp:quick-response'
    plugin_attrib = 'action_selected'

class slixmpp.plugins.xep_0439.stanza.Response(xml=None, parent=None)

    interfaces = {'label', 'value'}
    name = 'response'
    namespace = 'urn:xmpp:tmp:quick-response'
    plugin_attrib = 'response'

slixmpp.plugins.xep_0439.stanza.register_plugins()
```

XEP 0444

```
class slixmpp.plugins.xep_0444.XEP_0444(xmpp, config=None)
```

```
send_reactions(to, to_id, reactions, *, store=True)
```

Send reactions related to a message

```
static set_reactions(message, to_id, reactions)
```

Add reactions to a Message object.

```
stanza = <module 'slixmpp.plugins.xep_0444.stanza' from '/home/docs/checkouts/readthed
```

Stanza elements

Slixmpp: The Slick XMPP Library Copyright (C) 2020 Mathieu Pasquet This file is part of Slixmpp.

See the file LICENSE for copying permission.

```
class slixmpp.plugins.xep_0444.stanza.Reaction (xml=None, parent=None)

get_value()

    Return type str

interfaces = {'value'}
name = 'reaction'
namespace = 'urn:xmpp:reactions:0'
set_value(value, *, all_chars=False)

class slixmpp.plugins.xep_0444.stanza.Reactions (xml=None, parent=None)

get_values(*, all_chars=False)
    "Get all reactions as str

    Return type Set[str]

interfaces = {'id', 'values'}
name = 'reactions'
namespace = 'urn:xmpp:reactions:0'
plugin_attrib = 'reactions'
set_values(values, *, all_chars=False)
    "Set all reactions as str
```

6.13 Core Stanzas

6.13.1 Root Stanza

```
class slixmpp.stanza.rootstanza.RootStanza (stream=None, xml=None, stype=None,
                                              sto=None, sfrom=None, sid=None, parent=None)
```

A top-level XMPP stanza in an XMLStream.

The RootStanza class provides a more XMPP specific exception handler than provided by the generic StanzaBase class.

Methods: exception – Overrides StanzaBase.exception

exception(e)

Create and send an error reply.

Typically called when an event handler raises an exception. The error's type and text content are based on the exception object's type and content.

Overrides StanzaBase.exception.

Arguments: e – Exception object

6.13.2 Message Stanza

```
class slixmpp.stanza.Message(*args, **kwargs)
```

XMPP's <message> stanzas are a “push” mechanism to send information to other XMPP entities without requiring a response.

Chat clients will typically use <message> stanzas that have a type of either “chat” or “groupchat”.

When handling a message event, be sure to check if the message is an error response.

Example <message> stanzas:

```
<message to="user1@example.com" from="user2@example.com">
    <body>Hi!</body>
</message>

<message type="groupchat" to="room@conference.example.com">
    <body>Hi everyone!</body>
</message>
```

Stanza Interface:

- **body**: The main contents of the message.
- **subject**: An optional description of the message’s contents.
- **mucroom**: (Read-only) The name of the MUC room that sent the message.
- **mucnick**: (Read-only) The MUC nickname of message’s sender.

Attributes:

- **types**: May be one of: normal, chat, headline, groupchat, or error.

```
chat()
```

Set the message type to ‘chat’.

```
del_mucnick()
```

Dummy method to prevent deletion.

```
del_mucroom()
```

Dummy method to prevent deletion.

```
del_parent_thread()
```

Delete the message thread’s parent reference.

```
get_mucnick()
```

Return the nickname of the MUC user that sent the message.

Read-only stanza interface.

Return type str

```
get_mucroom()
```

Return the name of the MUC room where the message originated.

Read-only stanza interface.

Return type str

```
get_parent_thread()
```

Return the message thread’s parent thread.

Return type str

get_type()

Return the message type.

Overrides default stanza interface behavior.

Returns ‘normal’ if no type attribute is present.

Return type str

normal()

Set the message type to ‘normal’.

reply(*body=None, clear=True*)

Create a message reply.

Overrides StanzaBase.reply.

Sets proper ‘to’ attribute if the message is from a MUC, and adds a message body if one is given.

Parameters

- **body** (str) – Optional text content for the message.
- **clear** (bool) – Indicates if existing content should be removed before replying. Defaults to True.

Return type Message

set_mucnick(*value*)

Dummy method to prevent modification.

set_mucroom(*value*)

Dummy method to prevent modification.

set_parent_thread(*value*)

Add or change the message thread’s parent thread.

Parameters value (str) – identifier of the thread

6.13.3 Presence Stanza

class slixmpp.stanza.Presence(*args, **kwargs)

XMPP’s <presence> stanza allows entities to know the status of other clients and components. Since it is currently the only multi-cast stanza in XMPP, many extensions add more information to <presence> stanzas to broadcast to every entry in the roster, such as capabilities, music choices, or locations (XEP-0115: Entity Capabilities and XEP-0163: Personal Eventing Protocol).

Since <presence> stanzas are broadcast when an XMPP entity changes its status, the bulk of the traffic in an XMPP network will be from <presence> stanzas. Therefore, do not include more information than necessary in a status message or within a <presence> stanza in order to help keep the network running smoothly.

Example <presence> stanzas:

```
<presence />

<presence from="user@example.com">
    <show>away</show>
    <status>Getting lunch.</status>
    <priority>5</priority>
</presence>

<presence type="unavailable" />
```

(continues on next page)

(continued from previous page)

```
<presence to="user@otherhost.com" type="subscribe" />
```

Stanza Interface:

- **priority**: A value used by servers to determine message routing.
- **show**: The type of status, such as away or available for chat.
- **status**: Custom, human readable status message.

Attributes:

- **types**: One of: available, unavailable, error, probe, subscribe, subscribed, unsubscribe, and unsubscribed.
- **showtypes**: One of: away, chat, dnd, and xa.

del_type ()

Remove both the type attribute and the <show> element.

get_priority ()

Return the value of the <presence> element as an integer.

Return type int

get_type ()

Return the value of the <presence> stanza's type attribute, or the value of the <show> element if valid.

reply (clear=True)

Create a new reply <presence/> stanza from self.

Overrides StanzaBase.reply.

Parameters **clear** (bool) – Indicates if the stanza contents should be removed before replying. Defaults to True.

set_priority (value)

Set the entity's priority value. Some server use priority to determine message routing behavior.

Bot clients should typically use a priority of 0 if the same JID is used elsewhere by a human-interacting client.

Parameters **value** (int) – An integer value greater than or equal to 0.

set_show (show)

Set the value of the <show> element.

Parameters **show** (str) – Must be one of: away, chat, dnd, or xa.

set_type (value)

Set the type attribute's value, and the <show> element if applicable.

Parameters **value** (str) – Must be in either self.types or self.showtypes.

6.13.4 IQ Stanza

```
class slixmpp.stanza.Iq(*args, **kwargs)
```

XMPP <iq> stanzas, or info/query stanzas, are XMPP's method of requesting and modifying information, similar to HTTP's GET and POST methods.

Each <iq> stanza must have an 'id' value which associates the stanza with the response stanza. XMPP entities must always be given a response <iq> stanza with a type of 'result' after sending a stanza of type 'get' or 'set'.

Most uses cases for <iq> stanzas will involve adding a <query> element whose namespace indicates the type of information desired. However, some custom XMPP applications use <iq> stanzas as a carrier stanza for an application-specific protocol instead.

Example <iq> Stanzas:

```
<iq to="user@example.com" type="get" id="314">
    <query xmlns="http://jabber.org/protocol/disco#items" />
</iq>

<iq to="user@localhost" type="result" id="17">
    <query xmlns='jabber:iq:roster'>
        <item jid='otheruser@example.net'
              name='John Doe'
              subscription='both'>
            <group>Friends</group>
        </item>
    </query>
</iq>
```

Stanza Interface:

- **query**: The namespace of the <query> element if one exists.

Attributes:

- **types**: May be one of: get, set, result, or error.

del_query()

Remove the <query> element.

get_query()

Return the namespace of the <query> element.

Return type str

reply(*clear=True*)

Create a new <iq> stanza replying to self.

Overrides StanzaBase.reply

Sets the 'type' to 'result' in addition to the default StanzaBase.reply behavior.

Parameters **clear** (*bool*) – Indicates if existing content should be removed before replying.

Defaults to True.

send(*callback=None*, *timeout=None*, *timeout_callback=None*)

Send an <iq> stanza over the XML stream.

A callback handler can be provided that will be executed when the Iq stanza's result reply is received.

Returns a future which result will be set to the result Iq if it is of type 'get' or 'set' (when it is received), or a future with the result set to None if it has another type.

Overrides StanzaBase.send

Parameters

- **callback** (*function*) – Optional reference to a stream handler function. Will be executed when a reply stanza is received.
- **timeout** (*int*) – The length of time (in seconds) to wait for a response before the timeout_callback is called, instead of the regular callback
- **timeout_callback** (*function*) – Optional reference to a stream handler function. Will be executed when the timeout expires before a response has been received for the originally-sent IQ stanza.

Return type asyncio.Future

set_payload (*value*)

Set the XML contents of the <iq> stanza.

Parameters value (*list or XML object*) – An XML object or a list of XML objects to use as the <iq> stanza's contents

set_query (*value*)

Add or modify a <query> element.

Query elements are differentiated by their namespace.

Parameters value (*str*) – The namespace of the <query> element.

unhandled ()

Send a feature-not-implemented error if the stanza is not handled.

Overrides StanzaBase.unhandled.

ADDITIONAL INFO

7.1 Glossary

event handler A callback function that responds to events raised by `XMLStream.event()`.

interfaces A set of keys defined on a `stanza plugin`.

stanza An XML payload sent over the XML stream, which is the root of XMPP. A stanza is either `<iq/>`, `<message/>` or `<presence/>`. Other elements are called nonzas.

stanza object Informally may refer both to classes which extend `ElementBase` or `StanzaBase`, and to objects of such classes.

A stanza object is a wrapper for an XML object which exposes `dict` like interfaces which may be assigned to, read from, or deleted.

stanza plugin A `stanza object` which has been registered as a potential child of another stanza object. The plugin stanza may accessed through the parent stanza using the plugin's `plugin_attrib` as an interface.

stream handler A callback function that accepts stanza objects pulled directly from the XML stream. A stream handler is encapsulated in a object that includes a `Matcher` object which provides additional semantics.

substanza See `stanza plugin`

7.2 License (MIT)

Copyright (c) 2010 Nathanael C. Fritz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

(continues on next page)

(continued from previous page)

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

Licenses of Bundled Third Party Code

dateutil - Extensions to the standard python 2.3+ datetime module.

Copyright (c) 2003-2011 - Gustavo Niemeyer <gustavo@niemeyer.net>

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
- * Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

fixed_datetime

Copyright (c) 2008, Red Innovation Ltd., Finland
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of Red Innovation nor the names of its contributors
may be used to endorse or promote products derived from this software
without specific prior written permission.

(continues on next page)

(continued from previous page)

THIS SOFTWARE IS PROVIDED BY RED INNOVATION ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL RED INNOVATION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SUELTA - A PURE-PYTHON SASL CLIENT LIBRARY

This software is subject to "The MIT License"

Copyright 2004-2013 David Alan Cridland

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

python-gnupg: A Python wrapper for the GNU Privacy Guard

Copyright (c) 2008-2012 by Vinay Sajip.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name(s) of the copyright holder(s) may not be used to endorse or promote products derived from this software without specific prior written permission.

(continues on next page)

(continued from previous page)

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- [*License \(MIT\)*](#)
- [*Glossary*](#)
- [*genindex*](#)
- [*modindex*](#)
- [*search*](#)

CHAPTER
EIGHT

SLEEKXMPP CREDITS

Note: Those people made SleekXMPP, so you should not bother them if you have an issue with slixmpp. But it's still fair to credit them for their work.

Main Author: **Nathan Fritz** fritzy@netflint.net, [@fritzy](https://twitter.com/fritzy)

Nathan is also the author of XMPPHP and Seesmic-AS3-XMPP, and a former member of the XMPP Council.

Co-Author: **Lance Stout** lancestout@gmail.com, [@lancestout](https://twitter.com/lancestout)

Both Fritz and Lance work for [&yet](http://andyet.net), which specializes in realtime web and XMPP applications.

- contact@andyet.net
- [XMPP Consulting](#)

Contributors:

- Brian Beggs ([macdiesel](#))
- Dann Martens ([dannmartens](#))
- Florent Le Coz ([louiz](#))
- Kevin Smith ([Kev](#), <http://kismith.co.uk>)
- Remko Tronçon ([remko](#), <http://el-tramo.be>)
- Te-jé Rogers ([te-je](#))
- Thom Nichols ([tomstrummer](#))

PYTHON MODULE INDEX

S

slixmpp.basexmpp, 69
slixmpp.clientxmpp, 67
slixmpp.componentxmpp, 68
slixmpp.exceptions, 74
slixmpp.jid, 75
slixmpp.plugins.xep_0004, 100
slixmpp.plugins.xep_0004.stanza.field,
 100
slixmpp.plugins.xep_0004.stanza.form,
 102
slixmpp.plugins.xep_0009, 104
slixmpp.plugins.xep_0009.stanza.RPC, 104
slixmpp.plugins.xep_0012, 105
slixmpp.plugins.xep_0012.stanza, 105
slixmpp.plugins.xep_0013, 105
slixmpp.plugins.xep_0013.stanza, 106
slixmpp.plugins.xep_0020, 107
slixmpp.plugins.xep_0020.stanza, 107
slixmpp.plugins.xep_0027, 107
slixmpp.plugins.xep_0027.stanza, 107
slixmpp.plugins.xep_0030, 108
slixmpp.plugins.xep_0030.stanza.info,
 114
slixmpp.plugins.xep_0030.stanza.items,
 116
slixmpp.plugins.xep_0033, 118
slixmpp.plugins.xep_0033.stanza, 118
slixmpp.plugins.xep_0045, 120
slixmpp.plugins.xep_0045.stanza, 121
slixmpp.plugins.xep_0047, 125
slixmpp.plugins.xep_0047.stanza, 125
slixmpp.plugins.xep_0049, 126
slixmpp.plugins.xep_0049.stanza, 126
slixmpp.plugins.xep_0050, 126
slixmpp.plugins.xep_0050.stanza, 128
slixmpp.plugins.xep_0054, 130
slixmpp.plugins.xep_0054.stanza, 130
slixmpp.plugins.xep_0059, 139
slixmpp.plugins.xep_0059.stanza, 139
slixmpp.plugins.xep_0060, 140
slixmpp.plugins.xep_0060.stanza.base,
 143
slixmpp.plugins.xep_0060.stanza.pubsub,
 143
slixmpp.plugins.xep_0060.stanza.pubsub_errors,
 148
slixmpp.plugins.xep_0060.stanza.pubsub_event,
 151
slixmpp.plugins.xep_0060.stanza.pubsub_owner,
 148
slixmpp.plugins.xep_0065, 154
slixmpp.plugins.xep_0065.stanza, 155
slixmpp.plugins.xep_0066, 156
slixmpp.plugins.xep_0066.stanza, 156
slixmpp.plugins.xep_0070, 157
slixmpp.plugins.xep_0070.stanza, 157
slixmpp.plugins.xep_0071, 157
slixmpp.plugins.xep_0071.stanza, 157
slixmpp.plugins.xep_0077, 158
slixmpp.plugins.xep_0077.stanza, 158
slixmpp.plugins.xep_0079, 159
slixmpp.plugins.xep_0079.stanza, 159
slixmpp.plugins.xep_0080, 161
slixmpp.plugins.xep_0080.stanza, 162
slixmpp.plugins.xep_0082, 164
slixmpp.plugins.xep_0084, 164
slixmpp.plugins.xep_0084.stanza, 164
slixmpp.plugins.xep_0085, 165
slixmpp.plugins.xep_0085.stanza, 166
slixmpp.plugins.xep_0086, 167
slixmpp.plugins.xep_0086.stanza, 167
slixmpp.plugins.xep_0092, 168
slixmpp.plugins.xep_0092.stanza, 168
slixmpp.plugins.xep_0106, 169
slixmpp.plugins.xep_0107, 169
slixmpp.plugins.xep_0107.stanza, 169
slixmpp.plugins.xep_0108, 170
slixmpp.plugins.xep_0108.stanza, 170
slixmpp.plugins.xep_0115, 171
slixmpp.plugins.xep_0115.stanza, 171
slixmpp.plugins.xep_0118, 171
slixmpp.plugins.xep_0118.stanza, 172
slixmpp.plugins.xep_0122, 172

slixmpp.plugins.xep_0122.stanza, 172
slixmpp.plugins.xep_0128, 173
slixmpp.plugins.xep_0131, 174
slixmpp.plugins.xep_0131.stanza, 174
slixmpp.plugins.xep_0133, 174
slixmpp.plugins.xep_0152, 174
slixmpp.plugins.xep_0152.stanza, 175
slixmpp.plugins.xep_0153, 175
slixmpp.plugins.xep_0153.stanza, 175
slixmpp.plugins.xep_0163, 176
slixmpp.plugins.xep_0172, 177
slixmpp.plugins.xep_0172.stanza, 177
slixmpp.plugins.xep_0184, 178
slixmpp.plugins.xep_0184.stanza, 178
slixmpp.plugins.xep_0186, 179
slixmpp.plugins.xep_0186.stanza, 179
slixmpp.plugins.xep_0191, 180
slixmpp.plugins.xep_0191.stanza, 180
slixmpp.plugins.xep_0196, 180
slixmpp.plugins.xep_0196.stanza, 181
slixmpp.plugins.xep_0198, 181
slixmpp.plugins.xep_0198.stanza, 182
slixmpp.plugins.xep_0199, 184
slixmpp.plugins.xep_0199.stanza, 185
slixmpp.plugins.xep_0202, 185
slixmpp.plugins.xep_0202.stanza, 186
slixmpp.plugins.xep_0203, 187
slixmpp.plugins.xep_0203.stanza, 187
slixmpp.plugins.xep_0221, 187
slixmpp.plugins.xep_0221.stanza, 187
slixmpp.plugins.xep_0222, 188
slixmpp.plugins.xep_0223, 189
slixmpp.plugins.xep_0224, 190
slixmpp.plugins.xep_0224.stanza, 190
slixmpp.plugins.xep_0231, 191
slixmpp.plugins.xep_0231.stanza, 191
slixmpp.plugins.xep_0235, 191
slixmpp.plugins.xep_0235.stanza, 191
slixmpp.plugins.xep_0249, 192
slixmpp.plugins.xep_0249.stanza, 192
slixmpp.plugins.xep_0256, 193
slixmpp.plugins.xep_0257, 193
slixmpp.plugins.xep_0257.stanza, 193
slixmpp.plugins.xep_0258, 194
slixmpp.plugins.xep_0258.stanza, 194
slixmpp.plugins.xep_0279, 197
slixmpp.plugins.xep_0279.stanza, 197
slixmpp.plugins.xep_0280, 197
slixmpp.plugins.xep_0280.stanza, 197
slixmpp.plugins.xep_0297, 198
slixmpp.plugins.xep_0297.stanza, 199
slixmpp.plugins.xep_0300, 199
slixmpp.plugins.xep_0300.stanza, 199
slixmpp.plugins.xep_0308, 200
slixmpp.plugins.xep_0308.stanza, 200
slixmpp.plugins.xep_0313, 200
slixmpp.plugins.xep_0313.stanza, 201
slixmpp.plugins.xep_0319, 202
slixmpp.plugins.xep_0319.stanza, 202
slixmpp.plugins.xep_0332, 202
slixmpp.plugins.xep_0332.stanza.data, 203
slixmpp.plugins.xep_0332.stanza.request, 203
slixmpp.plugins.xep_0332.stanza.response, 204
slixmpp.plugins.xep_0333, 205
slixmpp.plugins.xep_0333.stanza, 205
slixmpp.plugins.xep_0334, 206
slixmpp.plugins.xep_0334.stanza, 206
slixmpp.plugins.xep_0335, 207
slixmpp.plugins.xep_0335.stanza, 207
slixmpp.plugins.xep_0352, 207
slixmpp.plugins.xep_0352.stanza, 207
slixmpp.plugins.xep_0353, 208
slixmpp.plugins.xep_0353.stanza, 208
slixmpp.plugins.xep_0359, 209
slixmpp.plugins.xep_0359.stanza, 209
slixmpp.plugins.xep_0363, 210
slixmpp.plugins.xep_0363.stanza, 210
slixmpp.plugins.xep_0369, 211
slixmpp.plugins.xep_0369.stanza, 213
slixmpp.plugins.xep_0377, 215
slixmpp.plugins.xep_0377.stanza, 215
slixmpp.plugins.xep_0380, 216
slixmpp.plugins.xep_0380.stanza, 216
slixmpp.plugins.xep_0394, 216
slixmpp.plugins.xep_0394.stanza, 216
slixmpp.plugins.xep_0403, 218
slixmpp.plugins.xep_0403.stanza, 218
slixmpp.plugins.xep_0404, 218
slixmpp.plugins.xep_0404.stanza, 219
slixmpp.plugins.xep_0405, 220
slixmpp.plugins.xep_0405.stanza, 220
slixmpp.plugins.xep_0421, 221
slixmpp.plugins.xep_0421.stanza, 221
slixmpp.plugins.xep_0422, 221
slixmpp.plugins.xep_0422.stanza, 222
slixmpp.plugins.xep_0424, 222
slixmpp.plugins.xep_0424.stanza, 223
slixmpp.plugins.xep_0425, 223
slixmpp.plugins.xep_0425.stanza, 223
slixmpp.plugins.xep_0428, 224
slixmpp.plugins.xep_0428.stanza, 224
slixmpp.plugins.xep_0437, 224
slixmpp.plugins.xep_0437.stanza, 224
slixmpp.plugins.xep_0439, 225
slixmpp.plugins.xep_0439.stanza, 226

slixmpp.plugins.xep_0444, 226
slixmpp.plugins.xep_0444.stanza, 227
slixmpp.stanza, 229
slixmpp.stanza.rootstanza, 227
slixmpp.xmlstream.handler, 89
slixmpp.xmlstream.handler.base, 88
slixmpp.xmlstream.matcher.base, 91
slixmpp.xmlstream.matcher.id, 92
slixmpp.xmlstream.matcher.stanzapath,
 92
slixmpp.xmlstream.matcher.xmlmask, 93
slixmpp.xmlstream.matcher.xpath, 92
slixmpp.xmlstream.stanzabase, 76
slixmpp.xmlstream.xmlstream, 93

INDEX

Symbols

<code>_bool__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 79	<code>_set_stanza_values()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 82
<code>_copy__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 79	<code>_set_sub_text()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 82
<code>_delitem__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 79	A
<code>_eq__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 79	<code>abort()</code> (<i>slixmpp.xmlstream.XMLStream method</i>), 93
<code>_getitem__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	<code>accept(class in slixmpp.plugins.xep_0353.stanza)</code> , 208 Ack (class in <i>slixmpp.plugins.xep_0198.stanza</i>), 182
<code>_init__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	<code>ack()</code> (<i>slixmpp.plugins.xep_0184.XEP_0184 method</i>), 178
<code>_iter__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	<code>Acknowledged(class in slixmpp.plugins.xep_0333.stanza)</code> , 205
<code>_len__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	Action (class in <i>slixmpp.plugins.xep_0439.stanza</i>), 226 actions (<i>slixmpp.plugins.xep_0013.stanza.Item attribute</i>), 106
<code>_ne__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	actions (<i>slixmpp.plugins.xep_0050.stanza.Command attribute</i>), 129
<code>_next__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	ActionSelected (class in <i>slixmpp.plugins.xep_0439.stanza</i>), 226
<code>_repr__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	activate() (<i>slixmpp.plugins.xep_0065.XEP_0065 method</i>), 154
<code>_setitem__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 80	Active (class in <i>slixmpp.plugins.xep_0085.stanza</i>), 166
<code>_str__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 81	Active (class in <i>slixmpp.plugins.xep_0352.stanza</i>), 207 Activity (class in <i>slixmpp.plugins.xep_0437.stanza</i>), 224
<code>_weakref__</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase attribute</i>), 81	add_address() (<i>slixmpp.plugins.xep_0033.stanza.Addresses method</i>), 118
<code>_del_attr__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 81	add_command() (<i>slixmpp.plugins.xep_0050.XEP_0050 method</i>), 126
<code>_del_sub__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 81	add_equivalent() (<i>slixmpp.plugins.xep_0258.stanza.SecurityLabel method</i>), 196
<code>_get_attr__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 81	add_event_handler()
<code>_get_stanza_values()</code>	(<i>slixmpp.xmlstream.XMLStream method</i>), 93
	add_extended_info()
<code>_get_sub_text__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 82	(<i>slixmpp.plugins.xep_0128.XEP_0128 method</i>), 173
<code>_set_attr__()</code> (<i>slixmpp.xmlstream.stanzabase.ElementBase method</i>), 82	add_feature() (<i>slixmpp.plugins.xep_0030.stanza.info.DiscoInfo method</i>), 114

add_feature() (*slixmpp.plugins.xep_0030.XEP_0030* address_types (*slixmpp.plugins.xep_0033.stanza.Address method*), 108
 add_field() (*slixmpp.plugins.xep_0004.stanza.form.Form* addresses (*class in slixmpp.plugins.xep_0033.stanza*), 118
 add_field() (*slixmpp.plugins.xep_0077.stanza.RegisterAffiliation* (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 143
 add_filter() (*slixmpp.xmlstream.xmlstream.XMLStream* Affiliations (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 143
 add_identity() (*slixmpp.plugins.xep_0030.stanza.info.DiscoInfo* slixmpp.plugins.xep_0060.stanza.pubsub), 143
 add_identity() (*slixmpp.plugins.xep_0030.XEP_0030Agent* (class in *slixmpp.plugins.xep_0054.stanza*), 130
 method), 109
 add_info() (*slixmpp.plugins.xep_0084.stanza.MetaData* attribute), 199
 method), 165
 AMP (class in *slixmpp.plugins.xep_0079.stanza*), 159
 add_interest() (*slixmpp.plugins.xep_0163.XEP_0163AMPFeature* (class in *slixmpp.plugins.xep_0079.stanza*), 159
 method), 176
 add_item() (*slixmpp.plugins.xep_0004.stanza.form.Form* api (*slixmpp.basexmpp.BaseXMPP* attribute), 69
 method), 102
 append() (*slixmpp.plugins.xep_0030.stanza.items.DiscoItems* method), 149
 method), 117
 append() (*slixmpp.plugins.xep_0030.XEP_0030* method), 151
 append() (*slixmpp.xmlstream.stanzabase.ElementBase* method), 83
 AppendCert (class in *slixmpp.plugins.xep_0422.stanza*), 222
 add_note() (*slixmpp.plugins.xep_0050.stanza.Command* slixmpp.plugins.xep_0257.stanza), 193
 method), 129
 appendxml() (*slixmpp.xmlstream.stanzabase.ElementBase* method), 225
 add_option() (*slixmpp.plugins.xep_0004.stanza.field.FormField* method), 83
 method), 101
 ApplyTo (class in *slixmpp.plugins.xep_0422.stanza*), 222
 add_orgunit() (*slixmpp.plugins.xep_0054.stanza.Org* ask_for_actions()
 method), 135
 add_pointer() (*slixmpp.plugins.xep_0084.stanza.MetaData* (slixmpp.plugins.xep_0439.XEP_0439
 method), 165
 add_reported() (*slixmpp.plugins.xep_0004.stanza.form.Form* for_response()
 method), 102
 (slixmpp.plugins.xep_0439.XEP_0439
 add_rule() (*slixmpp.plugins.xep_0079.stanza.AMP* method), 225
 method), 159
 attention, 59
 add_streamhost() (*slixmpp.plugins.xep_0065.stanza.SockSession* (class in *slixmpp.plugins.xep_0224.stanza*), 190
 method), 155
 add_uri() (*slixmpp.plugins.xep_0221.stanza.Media* auto_authorize() (*slixmpp.basexmpp.BaseXMPP*
 method), 187
 property), 69
 addField() (*slixmpp.plugins.xep_0004.stanza.form.Form* auto_subscribe() (*slixmpp.basexmpp.BaseXMPP*
 method), 102
 property), 69
 addOption() (*slixmpp.plugins.xep_0004.stanza.field.FormField* B
 method), 101
 addReported() (*slixmpp.plugins.xep_0004.stanza.form.Form* BaseHandler (class in *slixmpp.xmlstream.handler.base*), 88
 method), 102
 Address (class in *slixmpp.plugins.xep_0033.stanza*), 118
 Address (class in *slixmpp.plugins.xep_0054.stanza*), 130
 Address (class in *slixmpp.plugins.xep_0152.stanza*), 175
 address (*slixmpp.xmlstream.xmlstream.XMLStream* attribute), 94
 BitsOfBinary (class in *slixmpp.plugins.xep_0231.stanza*), 191
 Block (class in *slixmpp.plugins.xep_0191.stanza*), 180

BlockCode (*class in slixmpp.plugins.xep_0394.stanza*), 216
 BlockList (*class in slixmpp.plugins.xep_0191.stanza*), 180
 BlockQuote (*class in slixmpp.plugins.xep_0394.stanza*), 216
 bool_interfaces (*slixmpp.plugins.xep_0013.stanza.OfflineState* (*class in slixmpp.plugins.xep_0085.stanza*), attribute), 106
 bool_interfaces (*slixmpp.plugins.xep_0054.stanza.Address* state_active, 59 attribute), 130
 bool_interfaces (*slixmpp.plugins.xep_0054.stanza.Classification* state_gone, 60 attribute), 132
 bool_interfaces (*slixmpp.plugins.xep_0054.stanza.Email* state_paused, 60 attribute), 132
 bool_interfaces (*slixmpp.plugins.xep_0054.stanza.Label* method), 89 attribute), 133
 bool_interfaces (*slixmpp.plugins.xep_0054.stanza.Telephone* method), 91 attribute), 137
 bool_interfaces (*slixmpp.xmlstream.stanzabase.ElementBase* (slixmpp.plugins.xep_0405.XEP_0405 attribute), 83
 boundjid (*slixmpp.basexmpp.BaseXMPP* attribute), 69

C

ca_certs (*slixmpp.xmlstream.xmlstream.XMLStream* attribute), 94
 Callback (*class in slixmpp.xmlstream.handler*), 89
 can_create_channel () (slixmpp.plugins.xep_0369.XEP_0369 method), 211
 cancel () (slixmpp.plugins.xep_0004.stanza.form.Form method), 102
 cancel_command () (slixmpp.plugins.xep_0050.XEP_0050 method), 127
 cancel_config () (slixmpp.plugins.xep_0045.XEP_0045 method), 120
 cancel_connection_attempt () (slixmpp.xmlstream.xmlstream.XMLStream method), 94
 Capabilities (*class in slixmpp.plugins.xep_0115.stanza*), 171
 carbon_received, 59
 carbon_sent, 59
 CarbonDisable (*class in slixmpp.plugins.xep_0280.stanza*), 197
 CarbonEnable (*class in slixmpp.plugins.xep_0280.stanza*), 197
 Catalog (*class in slixmpp.plugins.xep_0258.stanza*), 194
 CatalogItem (*class in slixmpp.plugins.xep_0258.stanza*), 195
 Categories (*class in slixmpp.plugins.xep_0054.stanza*), 131
 certfile (*slixmpp.xmlstream.xmlstream.XMLStream* attribute), 94
 CertItem (*class in slixmpp.plugins.xep_0257.stanza*), 193
 Certs (*class in slixmpp.plugins.xep_0257.stanza*), 193
 changed_status, 59
 changed_subscription, 59
 chat () (*slixmpp.stanza.Message* method), 228
 ClientJoin (*class in slixmpp.plugins.xep_0405.stanza*), 220
 ClientLeave (*class in slixmpp.plugins.xep_0405.stanza*), 220
 ClientRoster (*class in slixmpp.basexmpp.BaseXMPP* attribute), 70
 ClientXMPP, 55, 57
 ClientXMPP (*class in slixmpp.clientxmpp*), 67
 Close (*class in slixmpp.plugins.xep_0047.stanza*), 125
 close () (slixmpp.plugins.xep_0065.XEP_0065 method), 154
 CodeType (*class in slixmpp.plugins.xep_0394.stanza*), 216
 command, 60
 Command (*class in slixmpp.plugins.xep_0050.stanza*), 128
 command_[action], 60
 complete_command () (slixmpp.plugins.xep_0050.XEP_0050 method), 127
 ComponentXMPP, 55, 57
 ComponentXMPP (*class in slixmpp.componentxmpp*), 68

Composing (*class in slixmpp.plugins.xep_0085.stanza*), `decline()` (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120
condition_ns (*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCategory.xep_0060.stanza.pubsub*),
attribute), 148
conditions (*slixmpp.plugins.xep_0060.stanza.pubsub_errorCategory.xep_0060.stanza.pubsub*),
attribute), 148
Configure (*class in slixmpp.plugins.xep_0060.stanza.pubsub*), in `default_domain` (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 95
`default_ns` (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 95
`default_port` (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 95
DefaultConfig (*class in slixmpp.plugins.xep_0060.stanza.pubsub_owner*),
148
`del_actions` () (*slixmpp.plugins.xep_0050.stanza.Command method*), 129
`del_activities` () (*slixmpp.plugins.xep_0437.stanza.RAI method*), 225
`del_addresses` () (*slixmpp.plugins.xep_0033.stanza.Addresses method*), 118
`del_affiliation` () (*slixmpp.plugins.xep_0045.stanza.MUCBase method*), 122
`del_all` () (*slixmpp.plugins.xep_0033.stanza.Addresses method*), 118
`del_attention` () (*slixmpp.plugins.xep_0224.stanza.Attention method*), 190
`del_bcc` () (*slixmpp.plugins.xep_0033.stanza.Addresses method*), 118
`del_binval` () (*slixmpp.plugins.xep_0054.stanza.BinVal method*), 131
`del_block_size` () (*slixmpp.plugins.xep_0047.stanza.Open method*), 125
`del_body` () (*slixmpp.plugins.xep_0071.stanza.XHTML_IM method*), 157
`del_carbon_received` () (*slixmpp.plugins.xep_0280.stanza.ReceivedCarbon method*), 198
`del_carbon_sent` () (*slixmpp.plugins.xep_0280.stanza.SentCarbon method*), 198
`del_categories` () (*slixmpp.plugins.xep_0054.stanza.Categories method*), 131
`del_cc` () (*slixmpp.plugins.xep_0033.stanza.Addresses method*), 118
`del_cert_management` () (*slixmpp.plugins.xep_0257.stanza.AppendCert method*), 193
`del_chat_state` () (*slixmpp.plugins.xep_0085.stanza.ChatState method*), 166
`del_condition` () (*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCategory.xep_0060.stanza.pubsub*),
method), 148
`del_data` () (*slixmpp.plugins.xep_0047.stanza.Data*), 154

D

Data (*class in slixmpp.plugins.xep_0047.stanza*), 125
Data (*class in slixmpp.plugins.xep_0084.stanza*), 164
`data_received` () (*slixmpp.xmlstream.xmlstream.XMLStream method*), 95
`deactivate` () (*slixmpp.plugins.xep_0065.XEP_0065 method*), 154
`del_chat_state` () (*slixmpp.plugins.xep_0085.stanza.ChatState method*), 166
`del_condition` () (*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCategory.xep_0060.stanza.pubsub*),
method), 148
`del_data` () (*slixmpp.plugins.xep_0047.stanza.Data*), 154

```

    method), 125
del_data () (slixmpp.plugins.xep_0231.stanza.BitsOfBinary      del_items () (slixmpp.plugins.xep_0030.XEP_0030
    method), 191                                         method), 110
del_descriptions ()                               del_items () (slixmpp.plugins.xep_0191.stanza.BlockList
    (slixmpp.plugins.xep_0353.stanza.Produce      method), 180
    method), 209
del_event_handler ()                           del_jid () (slixmpp.plugins.xep_0045.stanza.MUCBase
    (slixmpp.xmlstream.xmlstream.XMLStream      method), 122
    method), 95
del_extended_info ()                          del_lines () (slixmpp.plugins.xep_0054.stanza.Label
    (slixmpp.plugins.xep_0128.XEP_0128          method), 133
    method), 173
del_feature () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfolliti () (in module
    method), 115                                         slixmpp.plugins.xep_0033.stanza), 119
del_feature () (slixmpp.plugins.xep_0030.XEP_0030  del_nick () (slixmpp.plugins.xep_0045.stanza.MUCBase
    method), 109                                         method), 122
del_features () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfollifok () (slixmpp.plugins.xep_0172.stanza.UserNick
    method), 115                                         method), 177
del_features () (slixmpp.plugins.xep_0030.XEP_0030del_node_handler ()
    method), 109                                         (slixmpp.plugins.xep_0030.XEP_0030
del_fields () (slixmpp.plugins.xep_0004.stanza.form.Form      method), 110
    method), 102                                         del_noreply () (slixmpp.plugins.xep_0033.stanza.Addresses
del_fields () (slixmpp.plugins.xep_0077.stanza.Register      method), 118
    method), 158                                         del_notes () (slixmpp.plugins.xep_0050.stanza.Command
del_filter () (slixmpp.xmlstream.xmlstream.XMLStream      method), 129
    method), 95                                         del_number () (slixmpp.plugins.xep_0054.stanza.Telephone
del_first_index ()                           del_optional () (slixmpp.plugins.xep_0198.stanza.StreamManagement
    (slixmpp.plugins.xep_0059.stanza.Set method), 140                                         method), 184
                                         del_options () (slixmpp.plugins.xep_0004.stanza.field.FormField
del_headers () (slixmpp.plugins.xep_0131.stanza.Headers      method), 101
    method), 174
del_identities () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfollifos () (slixmpp.plugins.xep_0060.stanza.pubsub.Options
    method), 115                                         method), 145
del_identities () (slixmpp.plugins.xep_0030.XEP_0030del_orgunits () (slixmpp.plugins.xep_0054.stanza.Org
    method), 109                                         method), 135
del_identity () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfollifoent_thread () (slixmpp.stanza.Message
    method), 115                                         method), 228
del_identity () (slixmpp.plugins.xep_0030.XEP_0030del_payload () (slixmpp.plugins.xep_0060.stanza.pubsub.Item
    method), 110                                         method), 144
del_instructions ()                           del_payload () (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event
    (slixmpp.plugins.xep_0004.stanza.form.Form      method), 153
    method), 102                                         del_payload () (slixmpp.xmlstream.stanzabase.StanzaBase
del_ip_check () (slixmpp.plugins.xep_0279.stanza.IPCheck      method), 87
    method), 197                                         del_per_hop () (slixmpp.plugins.xep_0079.stanza.AMP
del_item () (slixmpp.plugins.xep_0030.stanza.items.DiscoItems  method), 159
    method), 117                                         del_publish_options ()
del_item () (slixmpp.plugins.xep_0030.XEP_0030            (slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions
    method), 110                                         method), 146
del_item_attr () (slixmpp.plugins.xep_0045.stanza.MDBaselquery () (slixmpp.stanza.Iq method), 231
    method), 122                                         del_receipt () (slixmpp.plugins.xep_0184.stanza.Received
del_items () (slixmpp.plugins.xep_0004.stanza.form.Form      method), 178
    method), 102                                         del_redirect () (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event
del_items () (slixmpp.plugins.xep_0030.stanza.items.DiscoItems  method), 152
    method), 117                                         del_replyroom () (slixmpp.plugins.xep_0033.stanza.Addresses

```

```

    method), 119
del_replyto() (slixmpp.plugins.xep_0033.stanza.Addresses     del_value() (slixmpp.plugins.xep_0258.stanza.DisplayMarking
    method), 119                                         method), 195
del_reported() (slixmpp.plugins.xep_0004.stanza.form.Form   del_value() (slixmpp.plugins.xep_0258.stanza.ESSLabel
    method), 102                                         method), 195
del_request_receipt() (slixmpp.plugins.xep_0184.stanza.Request del_value() (slixmpp.plugins.xep_0300.stanza.Hash
    method), 178                                         method), 199
del_required() (slixmpp.plugins.xep_0004.stanza.field.FormField del_value() (slixmpp.plugins.xep_0335.stanza.JSON_Container
    method), 101                                         method), 207
del_required() (slixmpp.plugins.xep_0060.stanza.base.Option(delFailingslixmpp.plugins.xep_0203.stanza), 187
    method), 143                                         delete_node() (slixmpp.plugins.xep_0060.XEP_0060
del_required() (slixmpp.plugins.xep_0198.stanza.StreamManagement method), 141
    method), 184                                         DeletedType          (class           in
del_results() (slixmpp.plugins.xep_0013.stanza.Offline        slixmpp.plugins.xep_0394.stanza), 216
    method), 106                                         delFields() (slixmpp.plugins.xep_0004.stanza.form.Form
del_results() (slixmpp.plugins.xep_0313.stanza.MAM            method), 102
    method), 201                                         delInstructions()
del_role() (slixmpp.plugins.xep_0045.stanza.MUCBase           (slixmpp.plugins.xep_0004.stanza.form.Form
    method), 122                                         method), 102
del_room() (slixmpp.plugins.xep_0045.stanza.MUCBase           delOptions() (slixmpp.plugins.xep_0004.stanza.field.FormField
    method), 122                                         method), 101
del_roster_item() (slixmpp.clientxmpp.ClientXMPP      delReported() (slixmpp.plugins.xep_0004.stanza.form.Form
    method), 67                                         method), 102
del_stanza() (slixmpp.plugins.xep_0297.stanza.Forwarded  delRequired() (slixmpp.plugins.xep_0004.stanza.field.FormField
    method), 199                                         method), 101
del_status() (slixmpp.plugins.xep_0012.stanza.LastActivity delValue() (slixmpp.plugins.xep_0004.stanza.field.FormField
    method), 105                                         method), 101
del_status_codes() (slixmpp.plugins.xep_0045.stanza.MUCBase dependencies (slixmpp.plugins.xep_0332.XEP_0332
    method), 122                                         attribute), 202
del_text() (slixmpp.plugins.xep_0203.stanza.Delay         Desc (class in slixmpp.plugins.xep_0054.stanza), 132
    method), 187                                         Destroy (class in slixmpp.plugins.xep_0369.stanza),
del_time() (slixmpp.plugins.xep_0202.stanza.EntityTime  213
    method), 186                                         destroy() (slixmpp.plugins.xep_0045.XEP_0045
del_to() (slixmpp.plugins.xep_0033.stanza.Addresses       destroy_channel()
    method), 119                                         (slixmpp.plugins.xep_0369.XEP_0369
del_type() (slixmpp.stanza.Presence method), 230      det_types() (slixmpp.plugins.xep_0394.stanza.Span
del_unsupported() (slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCondition (class           in
    method), 148                                         slixmpp.plugins.xep_0257.stanza), 194
del_users() (slixmpp.plugins.xep_0257.stanza.CertItem disco_info, 60
    method), 193                                         disco_items, 60
del_value() (slixmpp.plugins.xep_0004.stanza.field.FormField DiscoInfo          (class           in
    method), 101                                         slixmpp.plugins.xep_0030.stanza.info), 114
del_value() (slixmpp.plugins.xep_0084.stanza.Data      DiscoItem          (class           in
    method), 164                                         slixmpp.plugins.xep_0030.stanza.items),
del_value() (slixmpp.plugins.xep_0107.stanza.UserMood    116
    method), 169                                         DiscoItems          (class           in
del_value() (slixmpp.plugins.xep_0108.stanza.UserActivity disconnect() (slixmpp.xmlstream.xmlstream.XMLStream
    method), 170                                         method), 95
del_value() (slixmpp.plugins.xep_0221.stanza.URI      method), 95
    method), 188

```

disconnect_reason
 (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 96

disconnected, 60

disconnected(*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 96

disconnected() (*slixmpp.plugins.xep_0198.XEP_0198 method*), 181

discover_proxies()
 (*slixmpp.plugins.xep_0065.XEP_0065 method*), 154

Displayed (class in *slixmpp.plugins.xep_0333.stanza*), 205

DisplayMarking (class in *slixmpp.plugins.xep_0258.stanza*), 195

dns_answers (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 96

dns_service (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 96

E

ElementBase (class in *slixmpp.xmlstream.stanzabase*), 78

Email (class in *slixmpp.plugins.xep_0054.stanza*), 132

EmphasisType (class in *slixmpp.plugins.xep_0394.stanza*), 217

Enable (class in *slixmpp.plugins.xep_0198.stanza*), 182

enable() (*slixmpp.xmlstream.stanzabase.ElementBase method*), 83

Enabled (class in *slixmpp.plugins.xep_0198.stanza*), 182

Encrypted (class in *slixmpp.plugins.xep_0027.stanza*), 107

Encryption (class in *slixmpp.plugins.xep_0380.stanza*), 216

end_session_on_disconnect
 (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 96

entity_time, 61

EntityTime (class in *slixmpp.plugins.xep_0202.stanza*), 186

eof_received() (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

EquivalentLabel (class in *slixmpp.plugins.xep_0258.stanza*), 196

error() (*slixmpp.xmlstream.stanzabase.StanzaBase method*), 87

error_map (*slixmpp.plugins.xep_0086.stanza.LegacyError attribute*), 167

ESSLabel (class in *slixmpp.plugins.xep_0258.stanza*), 195

Event (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 151

event handler, 233

event () (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

event_handled() (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

EventAssociate (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 152

EventCollection (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 152

EventConfiguration (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 152

EventDelete (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 152

EventDisassociate (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 153

EventItem (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 153

EventItems (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 153

EventPurge (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 153

EventRetract (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 153

EventSubscription (class in *slixmpp.plugins.xep_0060.stanza.pubsub_event*), 154

exception() (*slixmpp.baseXmpp.BaseXMPP method*), 70

exception() (*slixmpp.plugins.xep_0080.stanza.Geoloc method*), 162

exception() (*slixmpp.stanza.rootstanza.RootStanza method*), 227

exception() (*slixmpp.xmlstream.stanzabase.StanzaBase method*), 87

exception() (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

External (class in *slixmpp.plugins.xep_0422.stanza*), 222

F

Failed (class in *slixmpp.plugins.xep_0198.stanza*), 183

failed_auth, 61

FailedRule (class in *slixmpp.plugins.xep_0079.stanza*), 159

FailedRules (class in *slixmpp.plugins.xep_0079.stanza*), 159

```

Fallback (class in slixmpp.plugins.xep_0428.stanza),
    224
FeatureNegotiation (class      in   get_affiliation_list()
    slixmpp.plugins.xep_0020.stanza), 107
field() (slixmpp.plugins.xep_0004.stanza.form.Form
    property), 102
field_types (slixmpp.plugins.xep_0004.stanza.field.FormField
    attribute), 101
FieldOption (class      in   get_all() (slixmpp.plugins.xep_0033.stanza.Addresses
    slixmpp.plugins.xep_0004.stanza.field), 100
Fin (class in slixmpp.plugins.xep_0313.stanza), 201
Form (class in slixmpp.plugins.xep_0004.stanza.form), 102
form_fields (slixmpp.plugins.xep_0077.stanza.Register
    attribute), 158
form_types (slixmpp.plugins.xep_0004.stanza.form.Form
    attribute), 102
format () (slixmpp.exceptions.XMPPError method), 74
FormField (class      in   get_always () (slixmpp.plugins.xep_0313.stanza.Preferences
    slixmpp.plugins.xep_0004.stanza.field), 101
FormValidation (class      in   get_anon_by_id() (slixmpp.plugins.xep_0404.XEP_0404
    slixmpp.plugins.xep_0122.stanza), 172
Forwarded (class in slixmpp.plugins.xep_0297.stanza),
    199
from_b64 () (in       module  get_anon_jid()
    slixmpp.plugins.xep_0047.stanza), 125
fulljid() (slixmpp.basexmpp.BaseXMPP property),
    70
(slixmpp.plugins.xep_0045.stanza.MUCBase
method), 122
get_affiliation_list()
(slixmpp.plugins.xep_0045.XEP_0045
method), 120
get_all() (slixmpp.plugins.xep_0033.stanza.Addresses
method), 119
get_alt () (slixmpp.plugins.xep_0080.stanza.Geoloc
method), 163
get_always () (slixmpp.plugins.xep_0313.stanza.Preferences
method), 201
get_anon_by_id() (slixmpp.plugins.xep_0404.XEP_0404
method), 218
get_anon_jid()
(slixmpp.plugins.xep_0404.XEP_0404
method), 218
get_anon_raw () (slixmpp.plugins.xep_0404.XEP_0404
method), 219
get_answer () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
get_attention () (slixmpp.plugins.xep_0224.stanza.Attention
method), 190
get_basic () (slixmpp.plugins.xep_0122.stanza.FormValidation
method), 172
get_bcc () (slixmpp.plugins.xep_0033.stanza.Addresses
method), 119
get_bday () (slixmpp.plugins.xep_0054.stanza.Birthday
method), 131
get_bearing () (slixmpp.plugins.xep_0080.stanza.Geoloc
method), 163
get_before () (slixmpp.plugins.xep_0059.stanza.Set
method), 140
get_bgcolor () (slixmpp.plugins.xep_0258.stanza.DisplayMarking
method), 195
get_binval () (slixmpp.plugins.xep_0054.stanza.BinVal
method), 131
get_block_size () (slixmpp.plugins.xep_0047.stanza.Open
method), 125
get_body () (slixmpp.plugins.xep_0071.stanza.XHTML_IM
method), 157
get_carbon_received()
(slixmpp.plugins.xep_0280.stanza.ReceivedCarbon
method), 198
get_carbon_sent()
(slixmpp.plugins.xep_0280.stanza.SentCarbon
method), 198
get_categories () (slixmpp.plugins.xep_0054.stanza.Categories
method), 131
get_cc () (slixmpp.plugins.xep_0033.stanza.Addresses
method), 119
get_cert_management()
(slixmpp.plugins.xep_0257.stanza.AppendCert
method), 193
get_channel_info()

```

(*slixmpp.plugins.xep_0369.XEP_0369 method*), 211
 get_chat_state() (*slixmpp.plugins.xep_0085.stanza.ChatState*) (*slixmpp.plugins.xep_0079.stanza.AMP method*), 159
 get_code() (*slixmpp.plugins.xep_0332.stanza.response.HTTPResponse*) (*slixmpp.plugins.xep_0203.stanza.Delay method*), 187
 get_commands() (*slixmpp.plugins.xep_0050.XEP_0050*) get_from() (*slixmpp.plugins.xep_0258.stanza.Catalog method*), 194
 get_condition() (*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorPublicationStanzaBase*)
method, 87
 get_config() (*slixmpp.plugins.xep_0060.stanza.pubsub_getter.DefaultConfig*) (*slixmpp.plugins.xep_0054.stanza.Name method*), 134
 get_data() (*slixmpp.plugins.xep_0047.stanza.Data*) get_h() (*slixmpp.plugins.xep_0198.stanza.Ack method*), 182
 get_data() (*slixmpp.plugins.xep_0231.stanza.BitsOfBinary*) get_h() (*slixmpp.plugins.xep_0198.stanza.Resume method*), 183
 get_data() (*slixmpp.plugins.xep_0332.stanza.data.HTTPData*) get_h() (*slixmpp.plugins.xep_0198.stanza.Resumed method*), 184
 get_default() (*slixmpp.plugins.xep_0258.stanza.Catalog*) get_headers() (*slixmpp.plugins.xep_0131.stanza.Headers method*), 174
 get_delivered() (*slixmpp.plugins.xep_0033.stanza.Address*) identities() (*slixmpp.plugins.xep_0030.stanza.info.DiscoInfo method*), 115
 get_desc() (*slixmpp.plugins.xep_0054.stanza.Desc*)
method, 132
 get_descriptions() (*slixmpp.plugins.xep_0353.stanza.Propose method*), 209
 get_dns_records() (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96
 get_encrypted() (*slixmpp.plugins.xep_0027.stanza.Encrypted*)
method, 107
 get_end() (*slixmpp.plugins.xep_0313.stanza.MAM*)
method, 201
 get_entity_time() (*slixmpp.plugins.xep_0202.XEP_0202 method*), 185
 get_error() (*slixmpp.plugins.xep_0080.stanza.Geoloc method*), 163
 get_expiry() (*slixmpp.plugins.xep_0060.stanza.pubsub_event.Event*)
method, 154
 get_family() (*slixmpp.plugins.xep_0054.stanza.Name method*), 134
 get_fault() (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse*)
method, 104
 get_features() (*slixmpp.plugins.xep_0030.stanza.info.DiscoInfo*)
method, 141
 get_fgcolor() (*slixmpp.plugins.xep_0258.stanza.DisplayMarking*)
method, 180
 get_fields() (*slixmpp.plugins.xep_0004.stanza.form.Form*)
method, 103
 get_fields() (*slixmpp.plugins.xep_0077.stanza.Register*)
method, 158
 get_first_index()
 (*slixmpp.plugins.xep_0059.stanza.Set*)
method, 140
 get_info() (*slixmpp.plugins.xep_0030.XEP_0030*)
method, 110
 get_info_from_domain() (*slixmpp.plugins.xep_0030.XEP_0030 method*), 111
 get_instructions() (*slixmpp.plugins.xep_0004.stanza.form.Form method*), 103
 get_item() (*slixmpp.plugins.xep_0060.XEP_0060*)
method, 141
 get_item_attr() (*slixmpp.plugins.xep_0045.stanza.MUCBase*)
method, 122
 get_item_ids() (*slixmpp.plugins.xep_0060.XEP_0060*)
method, 141
 get_items() (*slixmpp.plugins.xep_0004.stanza.form.Form*)
method, 117
 get_items() (*slixmpp.plugins.xep_0030.stanza.items.DiscoItems*)
method, 117
 get_items() (*slixmpp.plugins.xep_0030.XEP_0030*)
method, 111
 get_items() (*slixmpp.plugins.xep_0060.XEP_0060*)
method, 132
 get_jabberid() (*slixmpp.plugins.xep_0054.stanza.JabberID*)
method, 132
 get_jid() (*slixmpp.plugins.xep_0013.stanza.Item*)
method, 106
 get_jid() (*slixmpp.plugins.xep_0033.stanza.Address*)
method, 118

```

get_jid() (slixmpp.plugins.xep_0045.stanza.MUCActor.get_name () (slixmpp.plugins.xep_0030.stanza.items.DiscoItem
    method), 121
                                         method), 116
get_jid() (slixmpp.plugins.xep_0045.stanza.MUCBase.get_network_address ()
    method), 122
                                         (slixmpp.plugins.xep_0065.XEP_0065
get_jid() (slixmpp.plugins.xep_0045.stanza.MUCUserItem      method), 154
    method), 124
                                         get_never () (slixmpp.plugins.xep_0313.stanza.Preferences
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation   method), 201
    method), 143
                                         get_nick () (slixmpp.plugins.xep_0045.stanza.MUCBase
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.Options     method), 122
    method), 145
                                         get_nick () (slixmpp.plugins.xep_0172.stanza.UserNick
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe   method), 177
    method), 146
                                         getNickname () (slixmpp.plugins.xep_0054.stanza.Nickname
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.Subscription method), 134
    method), 147
                                         get_node () (slixmpp.plugins.xep_0030.stanza.items.DiscoItem
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe method), 116
    method), 147
                                         get_node_affiliations ()
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub.event.EventSDirectiveplugins.xep_0060.XEP_0060
    method), 154
                                         method), 141
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub_owner.getOwnerRedirectFig ())
    method), 150
                                         (slixmpp.plugins.xep_0060.XEP_0060
get_jid() (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubstitution
    method), 150
                                         get_node_subscriptions ()
get_jid() (slixmpp.plugins.xep_0065.stanza.StreamHost        (slixmpp.plugins.xep_0060.XEP_0060
    method), 155
                                         method), 141
get_jid() (slixmpp.plugins.xep_0065.stanza.StreamHost.UsedNodes () (slixmpp.plugins.xep_0060.XEP_0060
    method), 155
                                         method), 141
get_jid_property () (slixmpp.plugins.xep_0045.XEP_0045
    method), 120
                                         get_noreply () (slixmpp.plugins.xep_0033.stanza.Addresses
get_lat () (slixmpp.plugins.xep_0080.stanza.Geoloc
    method), 163
                                         method), 119
get_lines () (slixmpp.plugins.xep_0054.stanza.Label
    method), 133
                                         get_note () (slixmpp.plugins.xep_0054.stanza.Note
get_lon () (slixmpp.plugins.xep_0080.stanza.Geoloc
    method), 163
                                         method), 134
get_mailer () (slixmpp.plugins.xep_0054.stanza.Mailer
    method), 133
                                         get_notes () (slixmpp.plugins.xep_0050.stanza.Command
get_max_age () (slixmpp.plugins.xep_0231.stanza.BitsOfBinary
    method), 191
                                         method), 129
get_message () (slixmpp.plugins.xep_0332.stanza.response.HTTPMethodDelete101
    method), 204
                                         get_notify () (slixmpp.plugins.xep_0060.stanza.pubsub.Retract
get_method () (slixmpp.plugins.xep_0332.stanza.request.HTTPRequestMethod), 145
    method), 203
                                         method), 146
get_method_name () (slixmpp.plugins.xep_0009.stanza.RPC.MethodCall.get_our_jid_in_room()
    method), 104
                                         (slixmpp.plugins.xep_0045.XEP_0045
get_middle () (slixmpp.plugins.xep_0054.stanza.Name
    method), 134
                                         method), 120
get_mucnick () (slixmpp.stanza.Message method),
    228
                                         get_params () (slixmpp.plugins.xep_0009.stanza.RPC.MethodCall
get_mucroom () (slixmpp.stanza.Message method),
    228
                                         method), 104
get_multi () (in module
    slixmpp.plugins.xep_0033.stanza), 119
                                         get_parent_thread () (slixmpp.stanza.Message
                                         method), 228
                                         get_payload () (slixmpp.plugins.xep_0060.stanza.pubsub.Item

```

method), 144
get_payload () (slixmpp.plugins.xep_0060.stanza.pubsub.event.EventSet (slixmpp.plugins.xep_0313.stanza.MAM method), 153
get_payload () (slixmpp.xmlstream.stanzabase.StanzaBase.set_resume () (slixmpp.plugins.xep_0198.stanza.Enable method), 87
get_per_hop () (slixmpp.plugins.xep_0079.stanza.AMP get_resume () (slixmpp.plugins.xep_0198.stanza.Enabled method), 159
get_photo () (slixmpp.plugins.xep_0153.stanza.VCardTag.getUpdate () (slixmpp.plugins.xep_0054.stanza.Rev method), 175
get_preferences () (slixmpp.plugins.xep_0404.XEP_0404 method), 219
get_prefix () (slixmpp.plugins.xep_0054.stanza.Name method), 134
get_priority () (slixmpp.stanza.Presence method), 230
get_prodid () (slixmpp.plugins.xep_0054.stanza.ProdID method), 135
get_publish_options () (slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions method), 146
get_query () (slixmpp.stanza.Iq method), 231
get_range () (slixmpp.plugins.xep_0122.stanza.FormValidation roster () (slixmpp.plugins.xep_0045.XEP_0045 method), 172
get_receipt () (slixmpp.plugins.xep_0184.stanza.Receipt get_seconds () (slixmpp.plugins.xep_0012.stanza.LastActivity method), 178
get_redirect () (slixmpp.plugins.xep_0060.stanza.pubsub.event.EventDelete (slixmpp.plugins.xep_0047.stanza.Data method), 152
get_regex () (slixmpp.plugins.xep_0122.stanza.FormValidation signed () (slixmpp.plugins.xep_0027.stanza.Signed method), 172
get_registered () (slixmpp.plugins.xep_0077.stanza.Registered since () (slixmpp.plugins.xep_0319.stanza.Idle method), 158
get_remove () (slixmpp.plugins.xep_0077.stanza.Registered get_socket () (slixmpp.plugins.xep_0065.XEP_0065 method), 158
get_replyroom () (slixmpp.plugins.xep_0033.stanza.Addresses sort_string () (slixmpp.plugins.xep_0054.stanza.SortString method), 119
get_replyto () (slixmpp.plugins.xep_0033.stanza.Addresses method), 119
get_reported () (slixmpp.plugins.xep_0004.stanza.form.Form method), 103
get_request_receipt () (slixmpp.plugins.xep_0184.stanza.Request method), 178
get_required () (slixmpp.plugins.xep_0004.stanza.field.FormField method), 96
get_required () (slixmpp.plugins.xep_0060.stanza.base.OptionalSet method), 143
get_required () (slixmpp.plugins.xep_0198.stanza.StreamManager method), 184
get_resource () (slixmpp.plugins.xep_0332.stanza.request.HTTPRequest (slixmpp.xmlstream.stanzabase.ElementBase method), 204
get_restrict () (slixmpp.plugins.xep_0258.stanza.Catalog_start () (slixmpp.plugins.xep_0313.stanza.MAM method), 194
get_results () (slixmpp.plugins.xep_0013.stanza.Offline get_start () (slixmpp.plugins.xep_0394.stanza.List method), 106
get_role () (slixmpp.plugins.xep_0045.stanza.MUCBase method), 122
get_roles_list () (slixmpp.plugins.xep_0045.XEP_0045 method), 120
get_room () (slixmpp.plugins.xep_0045.stanza.MUCBase method), 122
get_room_config () (slixmpp.plugins.xep_0045.XEP_0045 method), 120
get_roster () (slixmpp.clientxmpp.ClientXMPP method), 67
get_stanza () (slixmpp.plugins.xep_0297.stanza.Forwarded method), 187
get_stanza_values () (slixmpp.plugins.xep_0203.stanza.Delay method), 83
get_stanza () (slixmpp.plugins.xep_0297.stanza.Forwarded method), 201
get_start () (slixmpp.plugins.xep_0394.stanza.List method), 136
get_update () (slixmpp.plugins.xep_0045.XEP_0045 method), 120
get_update () (slixmpp.plugins.xep_0045.XEP_0045 method), 105
get_update () (slixmpp.plugins.xep_0047.stanza.Data method), 125
get_update () (slixmpp.plugins.xep_0027.stanza.Signed method), 107
get_update () (slixmpp.plugins.xep_0319.stanza.Idle method), 202
get_update () (slixmpp.plugins.xep_0065.XEP_0065 method), 154
get_update () (slixmpp.plugins.xep_0054.stanza.SortString method), 136
get_update () (slixmpp.plugins.xep_0377.stanza.Report method), 215
get_update () (slixmpp.plugins.xep_0080.stanza.Geolocation method), 163
get_update () (slixmpp.xmlstream.XMLStream method), 163
get_update () (slixmpp.plugins.xep_0203.stanza.Delay method), 83
get_update () (slixmpp.plugins.xep_0297.stanza.Forwarded method), 201
get_update () (slixmpp.plugins.xep_0394.stanza.List method), 136

```

        method), 217
get_status () (slixmpp.plugins.xep_0012.stanza.LastActivity_value () (slixmpp.plugins.xep_0258.stanza.DisplayMarking
method), 105
get_status_codes ()                               method), 188
(slixmpp.plugins.xep_0045.stanza.MUCBase
method), 122
get_suffix () (slixmpp.plugins.xep_0054.stanza.Name
method), 134
get_text () (slixmpp.plugins.xep_0203.stanza.Delay
method), 187
get_time () (slixmpp.plugins.xep_0202.stanza.EntityTime
method), 186
get_timestamp () (slixmpp.plugins.xep_0080.stanza.Geoloc
method), 163
get_title () (slixmpp.plugins.xep_0054.stanza.Title
method), 137
get_to () (slixmpp.plugins.xep_0033.stanza.Addresses
method), 119
get_to () (slixmpp.plugins.xep_0079.stanza.AMP
method), 159
get_to () (slixmpp.plugins.xep_0258.stanza.Catalog
method), 194
get_to () (slixmpp.xmlstream.stanzabase.StanzaBase
method), 87
get_type () (slixmpp.plugins.xep_0060.stanza.pubsub.Default
method), 144
get_type () (slixmpp.stanza.Message method), 228
get_type () (slixmpp.stanza.Presence method), 230
get_types () (slixmpp.plugins.xep_0394.stanza.Span
method), 218
get_tz () (slixmpp.plugins.xep_0054.stanza.TimeZone
method), 137
get_tzo () (slixmpp.plugins.xep_0202.stanza.EntityTime
method), 186
get_uid () (slixmpp.plugins.xep_0054.stanza.UID
method), 138
get_unsupported ()                               getReported () (slixmpp.plugins.xep_0004.stanza.form.Form
(slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubError
method), 148
method), 103
get_url () (slixmpp.plugins.xep_0054.stanza.URL
method), 138
get_users () (slixmpp.plugins.xep_0257.stanza.CertItem
method), 193
get_utc () (slixmpp.plugins.xep_0202.stanza.EntityTime
method), 186
get_value () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
get_value () (slixmpp.plugins.xep_0084.stanza.Data
method), 164
get_value () (slixmpp.plugins.xep_0107.stanza.UserMood
method), 169
get_value () (slixmpp.plugins.xep_0108.stanza.UserActivity
method), 170
get_value () (slixmpp.plugins.xep_0221.stanza.URI
method), 195
get_value () (slixmpp.plugins.xep_0258.stanza.ESSLabel
method), 195
get_value () (slixmpp.plugins.xep_0300.stanza.Hash
method), 199
get_value () (slixmpp.plugins.xep_0335.stanza.JSON_Container
method), 207
get_value () (slixmpp.plugins.xep_0363.stanza.Header
method), 210
get_value () (slixmpp.plugins.xep_0444.stanza.Reaction
method), 227
get_values () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 103
get_values () (slixmpp.plugins.xep_0444.stanza.Reactions
method), 227
get_version () (slixmpp.plugins.xep_0092.XEP_0092
method), 168
get_version () (slixmpp.plugins.xep_0332.stanza.request.HTTPRequest
method), 204
get_with () (slixmpp.plugins.xep_0313.stanza.MAM
method), 201
getAnswer () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
getFields () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 102
getInstructions ()                               getOptions () (slixmpp.plugins.xep_0004.stanza.field.FormField
(slixmpp.plugins.xep_0004.stanza.form.Form
method), 102
method), 101
getReported () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 103
getRequired () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
getType () (slixmpp.plugins.xep_0060.stanza.pubsub.Configure
method), 101
getValue () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
getValues () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 103
gmail_messages, 61
gmail_notify, 61
Gone (class in slixmpp.plugins.xep_0085.stanza), 166
got_offline, 61
got_online, 61
groupchat_direct_invite, 61
groupchat_invite, 61
groupchat_message, 61
groupchat_presence, 61
groupchat_subject, 62
upchat_subj, 62

```

H

handle_config_change()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_decline()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_error_message()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_invite()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_join()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_message()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_presence()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 120

handle_groupchat_subject()
 (*slixmpp.plugins.xep_0045.XEP_0045 method*), 121

handshake() (*slixmpp.plugins.xep_0065.XEP_0065 method*), 154

has_identity() (*slixmpp.plugins.xep_0030.XEP_0030 method*), 111

Hash (*class in slixmpp.plugins.xep_0300.stanza*), 199

Header (*class in slixmpp.plugins.xep_0363.stanza*), 210

Headers (*class in slixmpp.plugins.xep_0131.stanza*), 174

HTTPData (*class in slixmpp.plugins.xep_0332.stanza.data*), 203

HTTPRequest (*class in slixmpp.plugins.xep_0332.stanza.request*), 203

HTTPResponse (*class in slixmpp.plugins.xep_0332.stanza.response*), 204

I

ibb_stream_data, 62

ibb_stream_end, 62

ibb_stream_start, 62

Idle (*class in slixmpp.plugins.xep_0319.stanza*), 202

Inactive (*class in slixmpp.plugins.xep_0085.stanza*), 166

Inactive (*class in slixmpp.plugins.xep_0352.stanza*), 208

incoming_filter()
 (*slixmpp.componentxmpp.ComponentXMPP method*), 69

incoming_filter()
 (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

Info (*class in slixmpp.plugins.xep_0084.stanza*), 164

init_parser() (*slixmpp.xmlstream.xmlstream.XMLStream method*), 96

init_plugin() (*slixmpp.xmlstream.stanzabase.ElementBase method*), 84

interface (*slixmpp.plugins.xep_0421.stanza.OccupantId attribute*), 221

interfaces, 233

interfaces (*slixmpp.plugins.xep_0004.stanza.field.FieldOption attribute*), 100

interfaces (*slixmpp.plugins.xep_0004.stanza.field.FormField attribute*), 101

interfaces (*slixmpp.plugins.xep_0004.stanza.form.Form attribute*), 103

interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.MethodCall attribute*), 104

interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse attribute*), 104

interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.RPCQuery attribute*), 104

interfaces (*slixmpp.plugins.xep_0012.stanza.LastActivity attribute*), 105

interfaces (*slixmpp.plugins.xep_0013.stanza.Item attribute*), 106

interfaces (*slixmpp.plugins.xep_0013.stanza.Offline attribute*), 106

interfaces (*slixmpp.plugins.xep_0020.stanza.FeatureNegotiation attribute*), 107

interfaces (*slixmpp.plugins.xep_0027.stanza.Encrypted attribute*), 107

interfaces (*slixmpp.plugins.xep_0027.stanza.Signed attribute*), 107

in interfaces (*slixmpp.plugins.xep_0030.stanza.info.DiscoInfo attribute*), 115

interfaces (*slixmpp.plugins.xep_0030.stanza.items.DiscoItem attribute*), 116

interfaces (*slixmpp.plugins.xep_0030.stanza.items.DiscoItems attribute*), 117

interfaces (*slixmpp.plugins.xep_0033.stanza.Address attribute*), 118

interfaces (*slixmpp.plugins.xep_0033.stanza.Addresses attribute*), 119

interfaces (*slixmpp.plugins.xep_0045.stanza.MUCActor attribute*), 121

interfaces (*slixmpp.plugins.xep_0045.stanza.MUCAdminItem attribute*), 122

interfaces (*slixmpp.plugins.xep_0045.stanza.MUCBase attribute*), 122

interfaces (*slixmpp.plugins.xep_0045.stanza.MUCDecline attribute*), 123

interfaces (*slixmpp.plugins.xep_0045.stanza.MUCHistory attribute*), 123

attribute), 123

interfaces (slixmpp.plugins.xep_0045.stanza.MUCInvite attribute), 123

interfaces (slixmpp.plugins.xep_0045.stanza.MUCJoin attribute), 123

interfaces (slixmpp.plugins.xep_0045.stanza.MUCOwner attribute), 124

interfaces (slixmpp.plugins.xep_0045.stanza.MUCStatistics attribute), 124

interfaces (slixmpp.plugins.xep_0045.stanza.MUCUserItem attribute), 124

interfaces (slixmpp.plugins.xep_0047.stanza.Close attribute), 125

interfaces (slixmpp.plugins.xep_0047.stanza.Data attribute), 125

interfaces (slixmpp.plugins.xep_0047.stanza.Open attribute), 125

interfaces (slixmpp.plugins.xep_0049.stanza.PrivateXML attribute), 126

interfaces (slixmpp.plugins.xep_0050.stanza.Command attribute), 129

interfaces (slixmpp.plugins.xep_0054.stanza.Address attribute), 130

interfaces (slixmpp.plugins.xep_0054.stanza.Agent attribute), 130

interfaces (slixmpp.plugins.xep_0054.stanza.BinVal attribute), 131

interfaces (slixmpp.plugins.xep_0054.stanza.Birthday attribute), 131

interfaces (slixmpp.plugins.xep_0054.stanza.Categories attribute), 131

interfaces (slixmpp.plugins.xep_0054.stanza.Classification attribute), 132

interfaces (slixmpp.plugins.xep_0054.stanza.Desc attribute), 132

interfaces (slixmpp.plugins.xep_0054.stanza.Email attribute), 132

interfaces (slixmpp.plugins.xep_0054.stanza.Geo attribute), 132

interfaces (slixmpp.plugins.xep_0054.stanza.JabberID attribute), 132

interfaces (slixmpp.plugins.xep_0054.stanza.Label attribute), 133

interfaces (slixmpp.plugins.xep_0054.stanza.Logo attribute), 133

interfaces (slixmpp.plugins.xep_0054.stanza.Mailer attribute), 133

interfaces (slixmpp.plugins.xep_0054.stanza.Name attribute), 134

interfaces (slixmpp.plugins.xep_0054.stanza.Nickname attribute), 134

interfaces (slixmpp.plugins.xep_0054.stanza.Note attribute), 134

interfaces (slixmpp.plugins.xep_0054.stanza.Org attribute), 135

interfaces (slixmpp.plugins.xep_0054.stanza.Photo attribute), 135

interfaces (slixmpp.plugins.xep_0054.stanza.ProdID attribute), 135

interfaces (slixmpp.plugins.xep_0054.stanza.Rev attribute), 136

interfaces (slixmpp.plugins.xep_0054.stanza.Role attribute), 136

interfaces (slixmpp.plugins.xep_0054.stanza.SortString attribute), 136

interfaces (slixmpp.plugins.xep_0054.stanza.Sound attribute), 136

interfaces (slixmpp.plugins.xep_0054.stanza.Telephone attribute), 137

interfaces (slixmpp.plugins.xep_0054.stanza.TimeZone attribute), 137

interfaces (slixmpp.plugins.xep_0054.stanza.Title attribute), 137

interfaces (slixmpp.plugins.xep_0054.stanza.UID attribute), 138

interfaces (slixmpp.plugins.xep_0054.stanza.URL attribute), 138

interfaces (slixmpp.plugins.xep_0054.stanza.VCardTemp attribute), 138

interfaces (slixmpp.plugins.xep_0059.stanza.Set attribute), 140

interfaces (slixmpp.plugins.xep_0060.stanza.base.OptionalSetting attribute), 143

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation attribute), 143

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations attribute), 144

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Configure attribute), 144

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Create attribute), 144

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Default attribute), 144

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Item attribute), 144

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Items attribute), 145

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Options attribute), 145

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Publish attribute), 145

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions attribute), 146

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub attribute), 146

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Retract attribute), 146

interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute), 146

attribute), 146
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions (slixmpp.plugins.xep_0065.stanza.Socks5 attribute), 155
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions (slixmpp.plugins.xep_0065.stanza.StreamHost attribute), 155
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions (slixmpp.plugins.xep_0065.stanza.StreamHostUsed attribute), 155
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe (slixmpp.plugins.xep_0066.stanza.OOB attribute), 156
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_ErrorPubsub (slixmpp.plugins.xep_0066.stanza.OOBTransfer attribute), 156
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventInterfaces (slixmpp.plugins.xep_0070.stanza.Confirm attribute), 157
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventAssoc (slixmpp.plugins.xep_0071.stanza.XHTML_IM attribute), 157
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCollect (slixmpp.plugins.xep_0077.stanza.Register attribute), 158
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventConfig (slixmpp.plugins.xep_0077.stanza.RegisterFeature attribute), 158
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDelete (slixmpp.plugins.xep_0079.stanza.AMP attribute), 159
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDisassoc (slixmpp.plugins.xep_0079.stanza.Rule attribute), 160
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventEaten (slixmpp.plugins.xep_0080.stanza.Geoloc attribute), 163
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventEaten (slixmpp.plugins.xep_0084.stanza.Data attribute), 164
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventParse (slixmpp.plugins.xep_0084.stanza.Info attribute), 165
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventParse (slixmpp.plugins.xep_0084.stanza.MetaData attribute), 165
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventParse (slixmpp.plugins.xep_0084.stanza.Pointer attribute), 165
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultConf (slixmpp.plugins.xep_0085.stanza.ChatState attribute), 166
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultConf (slixmpp.plugins.xep_0086.stanza.LegacyError attribute), 167
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultConf (slixmpp.plugins.xep_0092.stanza.Version attribute), 168
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConf (slixmpp.plugins.xep_0107.stanza.UserMood attribute), 169
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultConf (slixmpp.plugins.xep_0108.stanza.UserActivity attribute), 170
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultDel (slixmpp.plugins.xep_0115.stanza.Capabilities attribute), 171
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefaultParse (slixmpp.plugins.xep_0118.stanza.UserTune attribute), 172
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedis (slixmpp.plugins.xep_0122.stanza.FormValidation attribute), 172
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedis (slixmpp.plugins.xep_0131.stanza.Headers attribute), 174
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedis (slixmpp.plugins.xep_0152.stanza.Address attribute), 175
interfaces (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Pubsub (slixmpp.plugins.xep_0152.stanza.Reachability attribute), 175

attribute), 175

interfaces (*slixmpp.plugins.xep_0153.stanza.VCardTempUpdate*) interfaces (*slixmpp.plugins.xep_0257.stanza.Certs attribute*), 193

interfaces (*slixmpp.plugins.xep_0172.stanza.UserNick*) interfaces (*slixmpp.plugins.xep_0257.stanza.DisableCert attribute*), 194

interfaces (*slixmpp.plugins.xep_0184.stanza.Received*) interfaces (*slixmpp.plugins.xep_0257.stanza.RevokeCert attribute*), 194

interfaces (*slixmpp.plugins.xep_0184.stanza.Request*) interfaces (*slixmpp.plugins.xep_0258.stanza.Catalog attribute*), 194

interfaces (*slixmpp.plugins.xep_0186.stanza.Invisible*) interfaces (*slixmpp.plugins.xep_0258.stanza.CatalogItem attribute*), 195

interfaces (*slixmpp.plugins.xep_0186.stanza.Visible*) interfaces (*slixmpp.plugins.xep_0258.stanza.DisplayMarking attribute*), 195

interfaces (*slixmpp.plugins.xep_0191.stanza.BlockList*) interfaces (*slixmpp.plugins.xep_0258.stanza.ESSLabel attribute*), 195

interfaces (*slixmpp.plugins.xep_0196.stanza.UserGame*) interfaces (*slixmpp.plugins.xep_0279.stanza.IPCheck attribute*), 197

interfaces (*slixmpp.plugins.xep_0198.stanza.Ack*) interfaces (*slixmpp.plugins.xep_0280.stanza.CarbonDisable attribute*), 197

interfaces (*slixmpp.plugins.xep_0198.stanza.Enable*) interfaces (*slixmpp.plugins.xep_0280.stanza.CarbonEnable attribute*), 197

interfaces (*slixmpp.plugins.xep_0198.stanza.Enabled*) interfaces (*slixmpp.plugins.xep_0280.stanza.PrivateCarbon attribute*), 198

interfaces (*slixmpp.plugins.xep_0198.stanza.Failed*) interfaces (*slixmpp.plugins.xep_0280.stanza.ReceivedCarbon attribute*), 198

interfaces (*slixmpp.plugins.xep_0198.stanza.RequestAck*) interfaces (*slixmpp.plugins.xep_0280.stanza.SentCarbon attribute*), 198

interfaces (*slixmpp.plugins.xep_0198.stanza.Resume*) interfaces (*slixmpp.plugins.xep_0297.stanza.Forwarded attribute*), 199

interfaces (*slixmpp.plugins.xep_0198.stanza.Resumed*) interfaces (*slixmpp.plugins.xep_0300.stanza.Hash attribute*), 199

interfaces (*slixmpp.plugins.xep_0198.stanza.StreamManagement*) interfaces (*slixmpp.plugins.xep_0308.stanza.Replace attribute*), 200

interfaces (*slixmpp.plugins.xep_0199.stanza.Ping*) interfaces (*slixmpp.plugins.xep_0313.stanza.MAM attribute*), 201

interfaces (*slixmpp.plugins.xep_0202.stanza.EntityTimers*) interfaces (*slixmpp.plugins.xep_0313.stanza.Preferences attribute*), 201

interfaces (*slixmpp.plugins.xep_0203.stanza.Delay*) interfaces (*slixmpp.plugins.xep_0313.stanza.Result attribute*), 202

interfaces (*slixmpp.plugins.xep_0221.stanza.Media*) interfaces (*slixmpp.plugins.xep_0319.stanza.Idle attribute*), 202

interfaces (*slixmpp.plugins.xep_0221.stanza.URI*) interfaces (*slixmpp.plugins.xep_0332.stanza.data.HTTPData attribute*), 203

interfaces (*slixmpp.plugins.xep_0224.stanza.Attention*) interfaces (*slixmpp.plugins.xep_0332.stanza.request.HTTPRequest attribute*), 204

interfaces (*slixmpp.plugins.xep_0231.stanza.BitsOfBinary*) interfaces (*slixmpp.plugins.xep_0332.stanza.response.HTTPResponse attribute*), 204

interfaces (*slixmpp.plugins.xep_0235.stanza.OAuth*) interfaces (*slixmpp.plugins.xep_0333.stanza.Acknowledged attribute*), 205

interfaces (*slixmpp.plugins.xep_0249.stanza.Invite*) interfaces (*slixmpp.plugins.xep_0333.stanza.Displayed attribute*), 205

interfaces (*slixmpp.plugins.xep_0257.stanza.AppendCert*) interfaces (*slixmpp.plugins.xep_0333.stanza.Received attribute*), 206

interfaces (*slixmpp.plugins.xep_0257.stanza.CertItem*) interfaces (*slixmpp.plugins.xep_0335.stanza.JSON_Container attribute*), 206

attribute), 207

interfaces (slixmpp.plugins.xep_0353.stanza.JingleMessage), 208

interfaces (slixmpp.plugins.xep_0353.stanza.Props), 209

interfaces (slixmpp.plugins.xep_0359.stanza.OriginID), 209

interfaces (slixmpp.plugins.xep_0359.stanza.StanzaID), 210

interfaces (slixmpp.plugins.xep_0363.stanza.Get attribute), 210

interfaces (slixmpp.plugins.xep_0363.stanza.Header attribute), 210

interfaces (slixmpp.plugins.xep_0363.stanza.Put attribute), 210

interfaces (slixmpp.plugins.xep_0363.stanza.Request attribute), 210

interfaces (slixmpp.plugins.xep_0369.stanza.Create attribute), 213

interfaces (slixmpp.plugins.xep_0369.stanza.Destroy attribute), 213

interfaces (slixmpp.plugins.xep_0369.stanza.Join attribute), 213

interfaces (slixmpp.plugins.xep_0369.stanza.MIX attribute), 213

interfaces (slixmpp.plugins.xep_0369.stanza.Participant), 213

interfaces (slixmpp.plugins.xep_0369.stanza.Subscribe attribute), 214

interfaces (slixmpp.plugins.xep_0369.stanza.Unsubscribe attribute), 214

interfaces (slixmpp.plugins.xep_0369.stanza.UpdateSubscription attribute), 214

interfaces (slixmpp.plugins.xep_0377.stanza.Report attribute), 215

interfaces (slixmpp.plugins.xep_0380.stanza.Encryption attribute), 216

interfaces (slixmpp.plugins.xep_0394.stanza.List attribute), 217

interfaces (slixmpp.plugins.xep_0394.stanza.Span attribute), 218

interfaces (slixmpp.plugins.xep_0403.stanza.MIXPresence attribute), 218

interfaces (slixmpp.plugins.xep_0404.stanza.Participant attribute), 219

interfaces (slixmpp.plugins.xep_0405.stanza.ClientJoin attribute), 220

interfaces (slixmpp.plugins.xep_0405.stanza.ClientLeave attribute), 220

interfaces (slixmpp.plugins.xep_0422.stanza.ApplyTo attribute), 220

attribute), 222

interfaces (slixmpp.plugins.xep_0422.stanza.External attribute), 222

interfaces (slixmpp.plugins.xep_0424.stanza.Retracted attribute), 223

interfaces (slixmpp.plugins.xep_0425.stanza.Moderate attribute), 223

interfaces (slixmpp.plugins.xep_0425.stanza.Moderated attribute), 223

interfaces (slixmpp.plugins.xep_0437.stanza.RAI attribute), 225

interfaces (slixmpp.plugins.xep_0439.stanza.Action attribute), 226

interfaces (slixmpp.plugins.xep_0439.stanza.ActionSelected attribute), 226

interfaces (slixmpp.plugins.xep_0439.stanza.Response attribute), 226

interfaces (slixmpp.plugins.xep_0444.stanza.Reaction attribute), 227

interfaces (slixmpp.plugins.xep_0444.stanza.Reactions attribute), 227

interfaces (slixmpp.xmlstream.stanzabase.ElementBase attribute), 84

InvalidRules (class in slixmpp.plugins.xep_0079.stanza), 160

Invisible (class in slixmpp.plugins.xep_0186.stanza), 179

Invite (class in slixmpp.plugins.xep_0249.stanza), 192

invite () (slixmpp.plugins.xep_0045.XEP_0045 method), 121

IPCheck (class in slixmpp.plugins.xep_0279.stanza), 197

Iq (class in slixmpp.stanza), 231

IqError (class in slixmpp.exceptions), 75

iq (slixmpp.exceptions.IqTimeout attribute), 75

Iq () (slixmpp.basexmpp.BaseXMPP method), 69

IqError, 75

is_component (slixmpp.basexmpp.BaseXMPP attribute), 70

is_extension (slixmpp.plugins.xep_0027.stanza.Encrypted attribute), 107

is_extension (slixmpp.plugins.xep_0027.stanza.Signed attribute), 107

is_extension (slixmpp.plugins.xep_0054.stanza.BinVal attribute), 131

is_extension (slixmpp.plugins.xep_0054.stanza.Birthday attribute), 131

is_extension (slixmpp.plugins.xep_0054.stanza.Categories attribute), 131

is_extension (slixmpp.plugins.xep_0054.stanza.Desc attribute), 132

is_extension (slixmpp.plugins.xep_0054.stanza.JabberID attribute), 133

```

is_extension(slixmpp.plugins.xep_0054.stanza.Mailer)  

    attribute), 133  

is_extension(slixmpp.plugins.xep_0054.stanza.Nickname)  

    attribute), 134  

is_extension(slixmpp.plugins.xep_0054.stanza.Note)  

    attribute), 134  

is_extension(slixmpp.plugins.xep_0054.stanza.ProdID)  

    attribute), 135  

is_extension (slixmpp.plugins.xep_0054.stanza.Rev)  

    attribute), 136  

is_extension(slixmpp.plugins.xep_0054.stanza.Role)  

    attribute), 136  

is_extension(slixmpp.plugins.xep_0054.stanza.SortString)  

    attribute), 136  

is_extension(slixmpp.plugins.xep_0054.stanza.TimeZone)  

    attribute), 137  

is_extension(slixmpp.plugins.xep_0054.stanza.Title)  

    attribute), 137  

is_extension(slixmpp.plugins.xep_0054.stanza.UID)  

    attribute), 138  

is_extension(slixmpp.plugins.xep_0054.stanza.URL)  

    attribute), 138  

is_extension(slixmpp.plugins.xep_0060.stanza.pubsub)  

    attribute), 146  

is_extension(slixmpp.plugins.xep_0085.stanza.ChatState)  

    attribute), 166  

is_extension(slixmpp.plugins.xep_0131.stanza.Headers)  

    attribute), 174  

is_extension(slixmpp.plugins.xep_0184.stanza.Received)  

    attribute), 178  

is_extension(slixmpp.plugins.xep_0184.stanza.Request)  

    attribute), 178  

is_extension(slixmpp.plugins.xep_0224.stanza.Attention)  

    attribute), 190  

is_extension(slixmpp.plugins.xep_0279.stanza.IPCheck)  

    attribute), 197  

is_extension(slixmpp.plugins.xep_0280.stanza.ReceivedCarbon)  

    attribute), 198  

is_extension(slixmpp.plugins.xep_0280.stanza.SentCarbon)  

    attribute), 198  

is_extension(slixmpp.plugins.xep_0332.stanza.data.HTTPData)  

    attribute), 203  

is_extension(slixmpp.xmlstream.stanzabase.ElementBase)  

    attribute), 84  

Item (class in slixmpp.plugins.xep_0013.stanza), 106  

Item (class in slixmpp.plugins.xep_0060.stanza.pubsub),  

    144  

Items (class in slixmpp.plugins.xep_0060.stanza.pubsub),  

    145  

iterables (slixmpp.xmlstream.stanzabase.ElementBase)  

    attribute), 84  

iterate() (slixmpp.plugins.xep_0059.XEP_0059)  

    method), 139  

JabberID (class in slixmpp.plugins.xep_0054.stanza),  

    132  

JID (class in slixmpp.jid), 75  

jid() (slixmpp.basexmpp.BaseXMPP property), 70  

jingle_message_accept, 62  

jingle_message_proceed, 62  

jingle_message_propose, 62  

jingle_message_reject, 62  

jingle_message_retract, 62  

JingleMessage (class in  

    slixmpp.plugins.xep_0353.stanza), 208  

Join (class in slixmpp.plugins.xep_0369.stanza), 213  

join_channel () (slixmpp.plugins.xep_0369.XEP_0369)  

    method), 211  

join_channel () (slixmpp.plugins.xep_0405.XEP_0405)  

    method), 220  

join_muc () (slixmpp.plugins.xep_0045.XEP_0045)  

    method), 121  

JSON_Container (class in  

    slixmpp.plugins.xep_0335.stanza), 207  

K  

publishOptions  

    keyfile (slixmpp.xmlstream.xmlstream.XMLStream at-  

    tribute), 97  

    keys () (slixmpp.xmlstream.stanzabase.ElementBase  

    method), 84  

    killed, 62  

L  

Label (class in slixmpp.plugins.xep_0054.stanza), 133  

Label (class in slixmpp.plugins.xep_0258.stanza), 196  

Lang_interfaces (slixmpp.plugins.xep_0030.info.DiscoInfo)  

    attribute), 115  

Lang_interfaces (slixmpp.plugins.xep_0071.stanza.XHTML_IM)  

    attribute), 157  

Lang_interfaces (slixmpp.plugins.xep_0152.stanza.Address)  

    attribute), 175  

Lang_interfaces (slixmpp.xmlstream.stanzabase.ElementBase)  

    attribute), 84  

last_activity, 62  

LastActivity (class in  

    slixmpp.plugins.xep_0012.stanza), 105  

Leave (class in slixmpp.plugins.xep_0369.stanza), 213  

leave_channel () (slixmpp.plugins.xep_0369.XEP_0369)  

    method), 212  

leave_channel () (slixmpp.plugins.xep_0405.XEP_0405)  

    method), 220  

leave_muc () (slixmpp.plugins.xep_0045.XEP_0045)  

    method), 121  

LegacyError (class in  

    slixmpp.plugins.xep_0086.stanza), 167  

Li (class in slixmpp.plugins.xep_0394.stanza), 217  

List (class in slixmpp.plugins.xep_0394.stanza), 217

```

list_channels () (*slixmpp.plugins.xep_0369.XEP_0369*)
 method), 212
 list_mix_nodes () (*slixmpp.plugins.xep_0369.XEP_0369*)
 method), 212
 list_participants ()
 (*slixmpp.plugins.xep_0369.XEP_0369*
 method), 212
 Logo (*class in slixmpp.plugins.xep_0054.stanza*), 133

M

Mailer (*class in slixmpp.plugins.xep_0054.stanza*), 133
 make_iq () (*slixmpp.basexmpp.BaseXMPP* method),
 70
 make_iq_error ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 70
 make_iq_get ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 70
 make_iq_query ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 71
 make_iq_result ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 71
 make_iq_set ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 71
 make_message ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 71
 make_presence ()
 (*slixmpp.basexmpp.BaseXMPP*
 method), 72
 make_query_roster ()
 (*slixmpp.basexmpp.BaseXMPP* method),
 72
 MAM (*class in slixmpp.plugins.xep_0313.stanza*), 201
 map_node_event () (*slixmpp.plugins.xep_0060.XEP_0060*)
 method), 141
 Markable (*class in slixmpp.plugins.xep_0333.stanza*),
 205
 marker, 63
 marker_acknowledged, 63
 marker_displayed, 63
 marker_received, 63
 Markup (*class in slixmpp.plugins.xep_0394.stanza*), 217
 match () (*slixmpp.xmlstream.handler.base.BaseHandler*
 method), 89
 match () (*slixmpp.xmlstream.matcher.base.MatcherBase*
 method), 91
 match ()
 (*slixmpp.xmlstream.matcher.id.MatcherId*
 method), 92
 match ()
 (*slixmpp.xmlstream.matcher.stanzapath.StanzaPath*
 method), 92
 match ()
 (*slixmpp.xmlstream.matcher.xmlmask.MatchXMLMask*
 method), 93
 match ()
 (*slixmpp.xmlstream.matcher.xpath.MatchXPath*
 method), 92
 match ()
 (*slixmpp.xmlstream.stanzabase.ElementBase*
 method), 84

MatcherBase
 (class
 in
 slixmpp.xmlstream.matcher.base), 91
 MatcherId
 (class
 in
 slixmpp.xmlstream.matcher.id), 92
 MatchXMLMask
 (class
 in
 slixmpp.xmlstream.matcher.xmlmask), 93
 MatchXPath
 (class
 in
 slixmpp.xmlstream.matcher.xpath), 92
 max_redirects
 (*slixmpp.basexmpp.BaseXMPP* at-
 tribute), 72
 Media (*class in slixmpp.plugins.xep_0221.stanza*), 187
 merge ()
 (*slixmpp.plugins.xep_0004.stanza.form.Form*
 method), 103
 message, 63
 Message (*class in slixmpp.stanza*), 228
 Message ()
 (*slixmpp.basexmpp.BaseXMPP* method),
 69
 message_correction, 63
 message_error, 63
 message_form, 63
 message_xform, 63
 MetaData (*class in slixmpp.plugins.xep_0084.stanza*),
 165
 MethodCall
 (class
 in
 slixmpp.plugins.xep_0009.stanza.RPC), 104
 MethodResponse
 (class
 in
 slixmpp.plugins.xep_0009.stanza.RPC), 104
 MIX (*class in slixmpp.plugins.xep_0369.stanza*), 213
 MIXPresence
 (class
 in
 slixmpp.plugins.xep_0403.stanza), 218
 Moderate (*class in slixmpp.plugins.xep_0425.stanza*),
 223
 Moderated (*class in slixmpp.plugins.xep_0425.stanza*),
 223
 module
 slixmpp.basexmpp, 69
 slixmpp.clientxmpp, 67
 slixmpp.componentxmpp, 68
 slixmpp.exceptions, 74
 slixmpp.jid, 75
 slixmpp.plugins.xep_0004, 100
 slixmpp.plugins.xep_0004.stanza.field,
 100
 slixmpp.plugins.xep_0004.stanza.form,
 102
 slixmpp.plugins.xep_0009, 104
 slixmpp.plugins.xep_0009.stanza.RPC,
 104
 slixmpp.plugins.xep_0012, 105
 slixmpp.plugins.xep_0012.stanza, 105
 slixmpp.plugins.xep_0013, 105
 slixmpp.plugins.xep_0013.stanza, 106
 slixmpp.plugins.xep_0020, 107
 slixmpp.plugins.xep_0020.stanza, 107
 slixmpp.plugins.xep_0027, 107

slixmpp.plugins.xep_0027.stanza, 107
slixmpp.plugins.xep_0030, 108
slixmpp.plugins.xep_0030.stanza.info, 114
slixmpp.plugins.xep_0030.stanza.items, 116
slixmpp.plugins.xep_0033, 118
slixmpp.plugins.xep_0033.stanza, 118
slixmpp.plugins.xep_0045, 120
slixmpp.plugins.xep_0045.stanza, 121
slixmpp.plugins.xep_0047, 125
slixmpp.plugins.xep_0047.stanza, 125
slixmpp.plugins.xep_0049, 126
slixmpp.plugins.xep_0049.stanza, 126
slixmpp.plugins.xep_0050, 126
slixmpp.plugins.xep_0050.stanza, 128
slixmpp.plugins.xep_0054, 130
slixmpp.plugins.xep_0054.stanza, 130
slixmpp.plugins.xep_0059, 139
slixmpp.plugins.xep_0059.stanza, 139
slixmpp.plugins.xep_0060, 140
slixmpp.plugins.xep_0060.stanza.base, 143
slixmpp.plugins.xep_0060.stanza.pubsub, 143
slixmpp.plugins.xep_0060.stanza.pubsub_error, 148
slixmpp.plugins.xep_0060.stanza.pubsub_error, 151
slixmpp.plugins.xep_0060.stanza.pubsub_error, 148
slixmpp.plugins.xep_0065, 154
slixmpp.plugins.xep_0065.stanza, 155
slixmpp.plugins.xep_0066, 156
slixmpp.plugins.xep_0066.stanza, 156
slixmpp.plugins.xep_0070, 157
slixmpp.plugins.xep_0070.stanza, 157
slixmpp.plugins.xep_0071, 157
slixmpp.plugins.xep_0071.stanza, 157
slixmpp.plugins.xep_0077, 158
slixmpp.plugins.xep_0077.stanza, 158
slixmpp.plugins.xep_0079, 159
slixmpp.plugins.xep_0079.stanza, 159
slixmpp.plugins.xep_0080, 161
slixmpp.plugins.xep_0080.stanza, 162
slixmpp.plugins.xep_0082, 164
slixmpp.plugins.xep_0084, 164
slixmpp.plugins.xep_0084.stanza, 164
slixmpp.plugins.xep_0085, 165
slixmpp.plugins.xep_0085.stanza, 166
slixmpp.plugins.xep_0086, 167
slixmpp.plugins.xep_0086.stanza, 167
slixmpp.plugins.xep_0092, 168
slixmpp.plugins.xep_0092.stanza, 168
slixmpp.plugins.xep_0106, 169
slixmpp.plugins.xep_0107, 169
slixmpp.plugins.xep_0107.stanza, 169
slixmpp.plugins.xep_0108, 170
slixmpp.plugins.xep_0108.stanza, 170
slixmpp.plugins.xep_0115, 171
slixmpp.plugins.xep_0115.stanza, 171
slixmpp.plugins.xep_0118, 171
slixmpp.plugins.xep_0118.stanza, 172
slixmpp.plugins.xep_0122, 172
slixmpp.plugins.xep_0122.stanza, 172
slixmpp.plugins.xep_0128, 173
slixmpp.plugins.xep_0131, 174
slixmpp.plugins.xep_0131.stanza, 174
slixmpp.plugins.xep_0133, 174
slixmpp.plugins.xep_0152, 174
slixmpp.plugins.xep_0152.stanza, 175
slixmpp.plugins.xep_0153, 175
slixmpp.plugins.xep_0153.stanza, 175
slixmpp.plugins.xep_0163, 176
slixmpp.plugins.xep_0172, 177
slixmpp.plugins.xep_0172.stanza, 177
slixmpp.plugins.xep_0184, 178
slixmpp.plugins.xep_0184.stanza, 178
slixmpp.plugins.xep_0186, 179
slixmpp.plugins.xep_0186.stanza, 179
slixmpp.plugins.xep_0191, 180
slixmpp.plugins.xep_0191.stanza, 180
slixmpp.plugins.xep_0196, 180
slixmpp.plugins.xep_0196.stanza, 181
slixmpp.plugins.xep_0198, 181
slixmpp.plugins.xep_0198.stanza, 182
slixmpp.plugins.xep_0199, 184
slixmpp.plugins.xep_0199.stanza, 185
slixmpp.plugins.xep_0202, 185
slixmpp.plugins.xep_0202.stanza, 186
slixmpp.plugins.xep_0203, 187
slixmpp.plugins.xep_0203.stanza, 187
slixmpp.plugins.xep_0221, 187
slixmpp.plugins.xep_0221.stanza, 187
slixmpp.plugins.xep_0222, 188
slixmpp.plugins.xep_0223, 189
slixmpp.plugins.xep_0224, 190
slixmpp.plugins.xep_0224.stanza, 190
slixmpp.plugins.xep_0231, 191
slixmpp.plugins.xep_0231.stanza, 191
slixmpp.plugins.xep_0235, 191
slixmpp.plugins.xep_0235.stanza, 191
slixmpp.plugins.xep_0249, 192
slixmpp.plugins.xep_0249.stanza, 192
slixmpp.plugins.xep_0256, 193
slixmpp.plugins.xep_0257, 193
slixmpp.plugins.xep_0257.stanza, 193
slixmpp.plugins.xep_0258, 194

slixmpp.plugins.xep_0258.stanza, 194
 slixmpp.plugins.xep_0279, 197
 slixmpp.plugins.xep_0279.stanza, 197
 slixmpp.plugins.xep_0280, 197
 slixmpp.plugins.xep_0280.stanza, 197
 slixmpp.plugins.xep_0297, 198
 slixmpp.plugins.xep_0297.stanza, 199
 slixmpp.plugins.xep_0300, 199
 slixmpp.plugins.xep_0300.stanza, 199
 slixmpp.plugins.xep_0308, 200
 slixmpp.plugins.xep_0308.stanza, 200
 slixmpp.plugins.xep_0313, 200
 slixmpp.plugins.xep_0313.stanza, 201
 slixmpp.plugins.xep_0319, 202
 slixmpp.plugins.xep_0319.stanza, 202
 slixmpp.plugins.xep_0332, 202
 slixmpp.plugins.xep_0332.stanza.data, 203
 slixmpp.plugins.xep_0332.stanza.request, slixmpp.xmlstream.matcher.stanzapath, 92
 slixmpp.plugins.xep_0332.stanza.responses, slixmpp.xmlstream.matcher.xmlmask, 93
 slixmpp.plugins.xep_0333, 205
 slixmpp.plugins.xep_0333.stanza, 205
 slixmpp.plugins.xep_0334, 206
 slixmpp.plugins.xep_0334.stanza, 206
 slixmpp.plugins.xep_0335, 207
 slixmpp.plugins.xep_0335.stanza, 207
 slixmpp.plugins.xep_0352, 207
 slixmpp.plugins.xep_0352.stanza, 207
 slixmpp.plugins.xep_0353, 208
 slixmpp.plugins.xep_0353.stanza, 208
 slixmpp.plugins.xep_0359, 209
 slixmpp.plugins.xep_0359.stanza, 209
 slixmpp.plugins.xep_0363, 210
 slixmpp.plugins.xep_0363.stanza, 210
 slixmpp.plugins.xep_0369, 211
 slixmpp.plugins.xep_0369.stanza, 213
 slixmpp.plugins.xep_0377, 215
 slixmpp.plugins.xep_0377.stanza, 215
 slixmpp.plugins.xep_0380, 216
 slixmpp.plugins.xep_0380.stanza, 216
 slixmpp.plugins.xep_0394, 216
 slixmpp.plugins.xep_0394.stanza, 216
 slixmpp.plugins.xep_0403, 218
 slixmpp.plugins.xep_0403.stanza, 218
 slixmpp.plugins.xep_0404, 218
 slixmpp.plugins.xep_0404.stanza, 219
 slixmpp.plugins.xep_0405, 220
 slixmpp.plugins.xep_0405.stanza, 220
 slixmpp.plugins.xep_0421, 221
 slixmpp.plugins.xep_0421.stanza, 221
 slixmpp.plugins.xep_0422, 221
 slixmpp.plugins.xep_0422.stanza, 222
 slixmpp.plugins.xep_0424, 222
 slixmpp.plugins.xep_0424.stanza, 223
 slixmpp.plugins.xep_0425, 223
 slixmpp.plugins.xep_0425.stanza, 223
 slixmpp.plugins.xep_0428, 224
 slixmpp.plugins.xep_0428.stanza, 224
 slixmpp.plugins.xep_0437, 224
 slixmpp.plugins.xep_0437.stanza, 224
 slixmpp.plugins.xep_0439, 225
 slixmpp.plugins.xep_0439.stanza, 226
 slixmpp.plugins.xep_0444, 226
 slixmpp.plugins.xep_0444.stanza, 227
 slixmpp.stanza, 229
 slixmpp.stanza.rootstanza, 227
 slixmpp.xmlstream.handler, 89
 slixmpp.xmlstream.handler.base, 88
 slixmpp.xmlstream.matcher.base, 91
 slixmpp.xmlstream.matcher.id, 92
 slixmpp.xmlstream.matcher.stanzapath,
 92
 muc::[room]::got_offline, 63
 muc::[room]::got_online, 63
 muc::[room]::message, 64
 muc::[room]::presence, 64
 MUCActor (class in slixmpp.plugins.xep_0045.stanza), 121
 MUCAdminItem (class in slixmpp.plugins.xep_0045.stanza), 122
 MUCAdminQuery (class in slixmpp.plugins.xep_0045.stanza), 122
 MUCBase (class in slixmpp.plugins.xep_0045.stanza), 122
 MUCDecline (class in slixmpp.plugins.xep_0045.stanza), 123
 MUCHistory (class in slixmpp.plugins.xep_0045.stanza), 123
 MUCInvite (class in slixmpp.plugins.xep_0045.stanza), 123
 MUCJoin (class in slixmpp.plugins.xep_0045.stanza), 123
 MUCMessage (class in slixmpp.plugins.xep_0045.stanza), 123
 MUCOwnerDestroy (class in slixmpp.plugins.xep_0045.stanza), 124
 MUCOwnerQuery (class in slixmpp.plugins.xep_0045.stanza), 124

MUCPresence	(class <i>slixmpp.plugins.xep_0045.stanza</i>), 124	in name (<i>slixmpp.plugins.xep_0045.stanza.MUCBase</i> attribute), 122
MUCStatus (class in <i>slixmpp.plugins.xep_0045.stanza</i>), 124		name (<i>slixmpp.plugins.xep_0045.stanza.MUCDecline</i> attribute), 123
MUCUserItem	(class <i>slixmpp.plugins.xep_0045.stanza</i>), 124	in name (<i>slixmpp.plugins.xep_0045.stanza.MUCHistory</i> attribute), 123
multi_line_types (<i>slixmpp.plugins.xep_0004.stanza.field.FormField</i> attribute), 101		name (<i>slixmpp.plugins.xep_0045.stanza.MUCInvite</i> attribute), 123
multi_value_types (<i>slixmpp.plugins.xep_0004.stanza.field.FormField</i> attribute), 101		name (<i>slixmpp.plugins.xep_0045.stanza.MUCJoin</i> attribute), 123
		name (<i>slixmpp.plugins.xep_0045.stanza.MUCOwnerDestroy</i> attribute), 124
		name (<i>slixmpp.plugins.xep_0045.stanza.MUCOwnerQuery</i> attribute), 124
N		
Name (class in <i>slixmpp.plugins.xep_0054.stanza</i>), 134		
name (<i>slixmpp.plugins.xep_0004.stanza.field.FieldOption</i> attribute), 100		name (<i>slixmpp.plugins.xep_0045.stanza.MUCStatus</i> attribute), 124
name (<i>slixmpp.plugins.xep_0004.stanza.field.FormField</i> attribute), 101		name (<i>slixmpp.plugins.xep_0045.stanza.MUCUserItem</i> attribute), 124
name (<i>slixmpp.plugins.xep_0004.stanza.form.Form</i> attribute), 103		name (<i>slixmpp.plugins.xep_0047.stanza.Close</i> attribute), 125
name (<i>slixmpp.plugins.xep_0009.stanza.RPC.MethodCall</i> attribute), 104		name (<i>slixmpp.plugins.xep_0047.stanza.Data</i> attribute), 125
name (<i>slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse</i> attribute), 104		name (<i>slixmpp.plugins.xep_0047.stanza.Open</i> attribute), 125
name (<i>slixmpp.plugins.xep_0009.stanza.RPC.RPCQuery</i> attribute), 105		name (<i>slixmpp.plugins.xep_0049.stanza.PrivateXML</i> attribute), 126
name (<i>slixmpp.plugins.xep_0012.stanza.LastActivity</i> attribute), 105		name (<i>slixmpp.plugins.xep_0050.stanza.Command</i> attribute), 129
name (<i>slixmpp.plugins.xep_0013.stanza.Item</i> attribute), 106		name (<i>slixmpp.plugins.xep_0054.stanza.Address</i> attribute), 130
name (<i>slixmpp.plugins.xep_0013.stanza.Offline</i> attribute), 106		name (<i>slixmpp.plugins.xep_0054.stanza.Agent</i> attribute), 130
name (<i>slixmpp.plugins.xep_0020.stanza.FeatureNegotiation</i> attribute), 107		name (<i>slixmpp.plugins.xep_0054.stanza.BinVal</i> attribute), 131
name (<i>slixmpp.plugins.xep_0027.stanza.Encrypted</i> attribute), 107		name (<i>slixmpp.plugins.xep_0054.stanza.Birthday</i> attribute), 131
name (<i>slixmpp.plugins.xep_0027.stanza.Signed</i> attribute), 108		name (<i>slixmpp.plugins.xep_0054.stanza.Categories</i> attribute), 131
name (<i>slixmpp.plugins.xep_0030.stanza.info.DiscoInfo</i> attribute), 115		name (<i>slixmpp.plugins.xep_0054.stanza.Classification</i> attribute), 132
name (<i>slixmpp.plugins.xep_0030.stanza.items.DiscoItem</i> attribute), 116		name (<i>slixmpp.plugins.xep_0054.stanza.Desc</i> attribute), 132
name (<i>slixmpp.plugins.xep_0030.stanza.items.DiscoItems</i> attribute), 117		name (<i>slixmpp.plugins.xep_0054.stanza.Email</i> attribute), 132
name (<i>slixmpp.plugins.xep_0033.stanza.Address</i> attribute), 118		name (<i>slixmpp.plugins.xep_0054.stanza.Geo</i> attribute), 132
name (<i>slixmpp.plugins.xep_0033.stanza.Addresses</i> attribute), 119		name (<i>slixmpp.plugins.xep_0054.stanza.JabberID</i> attribute), 133
name (<i>slixmpp.plugins.xep_0045.stanza.MUCActor</i> attribute), 121		name (<i>slixmpp.plugins.xep_0054.stanza.Label</i> attribute), 133
name (<i>slixmpp.plugins.xep_0045.stanza.MUCAdminItem</i> attribute), 122		name (<i>slixmpp.plugins.xep_0054.stanza.Logo</i> attribute), 133
name (<i>slixmpp.plugins.xep_0045.stanza.MUCAdminQuery</i> attribute), 122		name (<i>slixmpp.plugins.xep_0054.stanza.Mailer</i> attribute), 133

name (<i>slixmpp.plugins.xep_0054.stanza.Name</i> attribute), 134	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub</i> attribute), 146
name (<i>slixmpp.plugins.xep_0054.stanza.Nickname</i> attribute), 134	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Retract</i> attribute), 146
name (<i>slixmpp.plugins.xep_0054.stanza.Note</i> attribute), 134	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe</i> attribute), 146
name (<i>slixmpp.plugins.xep_0054.stanza.Org</i> attribute), 135	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions</i> attribute), 147
name (<i>slixmpp.plugins.xep_0054.stanza.Photo</i> attribute), 135	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Subscription</i> attribute), 147
name (<i>slixmpp.plugins.xep_0054.stanza.ProdID</i> attribute), 135	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Subscriptions</i> attribute), 147
name (<i>slixmpp.plugins.xep_0054.stanza.Rev</i> attribute), 136	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe</i> attribute), 148
name (<i>slixmpp.plugins.xep_0054.stanza.Role</i> attribute), 136	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.Event</i> attribute), 151
name (<i>slixmpp.plugins.xep_0054.stanza.SortString</i> attribute), 136	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventAssociate</i> attribute), 152
name (<i>slixmpp.plugins.xep_0054.stanza.Sound</i> attribute), 136	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCollection</i> attribute), 152
name (<i>slixmpp.plugins.xep_0054.stanza.Telephone</i> attribute), 137	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventConfiguration</i> attribute), 152
name (<i>slixmpp.plugins.xep_0054.stanza.TimeZone</i> attribute), 137	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDelete</i> attribute), 153
name (<i>slixmpp.plugins.xep_0054.stanza.Title</i> attribute), 137	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDisassociate</i> attribute), 153
name (<i>slixmpp.plugins.xep_0054.stanza.UID</i> attribute), 138	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItem</i> attribute), 153
name (<i>slixmpp.plugins.xep_0054.stanza.URL</i> attribute), 138	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItems</i> attribute), 153
name (<i>slixmpp.plugins.xep_0054.stanza.VCardTemp</i> attribute), 138	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventPurge</i> attribute), 153
name (<i>slixmpp.plugins.xep_0059.stanza.Set</i> attribute), 140	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventRetract</i> attribute), 154
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation</i> attribute), 143	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.EventSubscription</i> attribute), 154
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations</i> attribute), 144	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.DefaultConfig</i> attribute), 148
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Configure</i> attribute), 144	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConfigure</i> attribute), 149
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Create</i> attribute), 144	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDelete</i> attribute), 150
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Default</i> attribute), 144	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerPurge</i> attribute), 150
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Item</i> attribute), 145	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedirect</i> attribute), 150
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Items</i> attribute), 145	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscription</i> attribute), 150
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Options</i> attribute), 145	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscription</i> attribute), 151
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Publish</i> attribute), 145	name (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.PubsubOwner</i> attribute), 151
name (<i>slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions</i> attribute), 146	name (<i>slixmpp.plugins.xep_0065.stanza.Socks5</i> attribute), 155

name (<i>slixmpp.plugins.xep_0065.stanza.StreamHost attribute</i>), 155	name (<i>slixmpp.plugins.xep_0092.stanza.Version attribute</i>), 168
name (<i>slixmpp.plugins.xep_0065.stanza.StreamHostUsed attribute</i>), 155	name (<i>slixmpp.plugins.xep_0107.stanza.UserMood attribute</i>), 169
name (<i>slixmpp.plugins.xep_0066.stanza.OOB attribute</i>), 156	name (<i>slixmpp.plugins.xep_0108.stanza.UserActivity attribute</i>), 170
name (<i>slixmpp.plugins.xep_0066.stanza.OOBTransfer attribute</i>), 156	name (<i>slixmpp.plugins.xep_0115.stanza.Capabilities attribute</i>), 171
name (<i>slixmpp.plugins.xep_0070.stanza.Confirm attribute</i>), 157	name (<i>slixmpp.plugins.xep_0118.stanza.UserTune attribute</i>), 172
name (<i>slixmpp.plugins.xep_0071.stanza.XHTML_IM attribute</i>), 157	name (<i>slixmpp.plugins.xep_0122.stanza.FormValidation attribute</i>), 172
name (<i>slixmpp.plugins.xep_0077.stanza.Register attribute</i>), 158	name (<i>slixmpp.plugins.xep_0131.stanza.Headers attribute</i>), 174
name (<i>slixmpp.plugins.xep_0077.stanza.RegisterFeature attribute</i>), 158	name (<i>slixmpp.plugins.xep_0152.stanza.Address attribute</i>), 175
name (<i>slixmpp.plugins.xep_0079.stanza.AMP attribute</i>), 159	name (<i>slixmpp.plugins.xep_0152.stanza.Reachability attribute</i>), 175
name (<i>slixmpp.plugins.xep_0079.stanza.AMPFeature attribute</i>), 159	name (<i>slixmpp.plugins.xep_0153.stanza.VCardTempUpdate attribute</i>), 175
name (<i>slixmpp.plugins.xep_0079.stanza.FailedRules attribute</i>), 159	name (<i>slixmpp.plugins.xep_0172.stanza.UserNick attribute</i>), 177
name (<i>slixmpp.plugins.xep_0079.stanza.InvalidRules attribute</i>), 160	name (<i>slixmpp.plugins.xep_0184.stanza.Received attribute</i>), 178
name (<i>slixmpp.plugins.xep_0079.stanza.Rule attribute</i>), 160	name (<i>slixmpp.plugins.xep_0184.stanza.Request attribute</i>), 179
name (<i>slixmpp.plugins.xep_0079.stanza.UnsupportedActions attribute</i>), 160	name (<i>slixmpp.plugins.xep_0186.stanza.Invisible attribute</i>), 179
name (<i>slixmpp.plugins.xep_0079.stanza.UnsupportedConditions attribute</i>), 160	name (<i>slixmpp.plugins.xep_0186.stanza.Visible attribute</i>), 179
name (<i>slixmpp.plugins.xep_0080.stanza.Geoloc attribute</i>), 163	name (<i>slixmpp.plugins.xep_0191.stanza.Block attribute</i>), 180
name (<i>slixmpp.plugins.xep_0084.stanza.Data attribute</i>), 164	name (<i>slixmpp.plugins.xep_0191.stanza.BlockList attribute</i>), 180
name (<i>slixmpp.plugins.xep_0084.stanza.Info attribute</i>), 165	name (<i>slixmpp.plugins.xep_0191.stanza.Unblock attribute</i>), 180
name (<i>slixmpp.plugins.xep_0084.stanza.MetaData attribute</i>), 165	name (<i>slixmpp.plugins.xep_0196.stanza.UserGaming attribute</i>), 181
name (<i>slixmpp.plugins.xep_0084.stanza.Pointer attribute</i>), 165	name (<i>slixmpp.plugins.xep_0198.stanza.Ack attribute</i>), 182
name (<i>slixmpp.plugins.xep_0085.stanza.Active attribute</i>), 166	name (<i>slixmpp.plugins.xep_0198.stanza.Enable attribute</i>), 182
name (<i>slixmpp.plugins.xep_0085.stanza.ChatState attribute</i>), 166	name (<i>slixmpp.plugins.xep_0198.stanza.Enabled attribute</i>), 182
name (<i>slixmpp.plugins.xep_0085.stanza.Composing attribute</i>), 166	name (<i>slixmpp.plugins.xep_0198.stanza.Failed attribute</i>), 183
name (<i>slixmpp.plugins.xep_0085.stanza.Gone attribute</i>), 166	name (<i>slixmpp.plugins.xep_0198.stanza.RequestAck attribute</i>), 183
name (<i>slixmpp.plugins.xep_0085.stanza.Inactive attribute</i>), 166	name (<i>slixmpp.plugins.xep_0198.stanza.Resume attribute</i>), 183
name (<i>slixmpp.plugins.xep_0085.stanza.Paused attribute</i>), 167	name (<i>slixmpp.plugins.xep_0198.stanza.Resumed attribute</i>), 184
name (<i>slixmpp.plugins.xep_0086.stanza.LegacyError attribute</i>), 167	name (<i>slixmpp.plugins.xep_0198.stanza.StreamManagement attribute</i>), 184

name (<i>slixmpp.plugins.xep_0199.stanza.Ping</i> attribute), 185	name (<i>slixmpp.plugins.xep_0297.stanza.Forwarded</i> attribute), 199
name (<i>slixmpp.plugins.xep_0202.stanza.EntityTime</i> attribute), 186	name (<i>slixmpp.plugins.xep_0300.stanza.Hash</i> attribute), 199
name (<i>slixmpp.plugins.xep_0203.stanza.Delay</i> attribute), 187	name (<i>slixmpp.plugins.xep_0308.stanza.Replace</i> attribute), 200
name (<i>slixmpp.plugins.xep_0221.stanza.Media</i> attribute), 187	name (<i>slixmpp.plugins.xep_0313.stanza.Fin</i> attribute), 201
name (<i>slixmpp.plugins.xep_0221.stanza.URI</i> attribute), 188	name (<i>slixmpp.plugins.xep_0313.stanza.MAM</i> attribute), 201
name (<i>slixmpp.plugins.xep_0224.stanza.Attention</i> attribute), 190	name (<i>slixmpp.plugins.xep_0313.stanza.Preferences</i> attribute), 201
name (<i>slixmpp.plugins.xep_0231.stanza.BitsOfBinary</i> attribute), 191	name (<i>slixmpp.plugins.xep_0313.stanza.Result</i> attribute), 202
name (<i>slixmpp.plugins.xep_0235.stanza.OAuth</i> attribute), 191	name (<i>slixmpp.plugins.xep_0319.stanza.Idle</i> attribute), 202
name (<i>slixmpp.plugins.xep_0249.stanza.Invite</i> attribute), 192	name (<i>slixmpp.plugins.xep_0332.stanza.data.HTTPData</i> attribute), 203
name (<i>slixmpp.plugins.xep_0257.stanza.AppendCert</i> attribute), 193	name (<i>slixmpp.plugins.xep_0332.stanza.request.HTTPRequest</i> attribute), 204
name (<i>slixmpp.plugins.xep_0257.stanza.CertItem</i> attribute), 193	name (<i>slixmpp.plugins.xep_0332.stanza.response.HTTPResponse</i> attribute), 204
name (<i>slixmpp.plugins.xep_0257.stanza.Certs</i> attribute), 193	name (<i>slixmpp.plugins.xep_0333.stanza.Acknowledged</i> attribute), 205
name (<i>slixmpp.plugins.xep_0257.stanza.DisableCert</i> attribute), 194	name (<i>slixmpp.plugins.xep_0333.stanza.Displayed</i> attribute), 205
name (<i>slixmpp.plugins.xep_0257.stanza.RevokeCert</i> attribute), 194	name (<i>slixmpp.plugins.xep_0333.stanza.Markable</i> attribute), 205
name (<i>slixmpp.plugins.xep_0258.stanza.Catalog</i> attribute), 194	name (<i>slixmpp.plugins.xep_0333.stanza.Received</i> attribute), 206
name (<i>slixmpp.plugins.xep_0258.stanza.CatalogItem</i> attribute), 195	name (<i>slixmpp.plugins.xep_0334.stanza.NoCopy</i> attribute), 206
name (<i>slixmpp.plugins.xep_0258.stanza.DisplayMarking</i> attribute), 195	name (<i>slixmpp.plugins.xep_0334.stanza.NoPermanentStore</i> attribute), 206
name (<i>slixmpp.plugins.xep_0258.stanza.EquivalentLabel</i> attribute), 196	name (<i>slixmpp.plugins.xep_0334.stanza.NoStore</i> attribute), 206
name (<i>slixmpp.plugins.xep_0258.stanza.ESSLLabel</i> attribute), 196	name (<i>slixmpp.plugins.xep_0334.stanza.Store</i> attribute), 206
name (<i>slixmpp.plugins.xep_0258.stanza.Label</i> attribute), 196	name (<i>slixmpp.plugins.xep_0335.stanza.JSON_Container</i> attribute), 207
name (<i>slixmpp.plugins.xep_0258.stanza.SecurityLabel</i> attribute), 196	name (<i>slixmpp.plugins.xep_0352.stanza.Active</i> attribute), 207
name (<i>slixmpp.plugins.xep_0279.stanza.IPCheck</i> attribute), 197	name (<i>slixmpp.plugins.xep_0352.stanza.ClientStateIndication</i> attribute), 208
name (<i>slixmpp.plugins.xep_0280.stanza.CarbonDisable</i> attribute), 197	name (<i>slixmpp.plugins.xep_0352.stanza.Inactive</i> attribute), 208
name (<i>slixmpp.plugins.xep_0280.stanza.CarbonEnable</i> attribute), 197	name (<i>slixmpp.plugins.xep_0353.stanza.Accept</i> attribute), 208
name (<i>slixmpp.plugins.xep_0280.stanza.PrivateCarbon</i> attribute), 198	name (<i>slixmpp.plugins.xep_0353.stanza.Proceed</i> attribute), 209
name (<i>slixmpp.plugins.xep_0280.stanza.ReceivedCarbon</i> attribute), 198	name (<i>slixmpp.plugins.xep_0353.stanza.Propose</i> attribute), 209
name (<i>slixmpp.plugins.xep_0280.stanza.SentCarbon</i> attribute), 198	name (<i>slixmpp.plugins.xep_0353.stanza.Reject</i> attribute), 209

name (<i>slixmpp.plugins.xep_0353.stanza.Retract attribute</i>), 209	name (<i>slixmpp.plugins.xep_0394.stanza.List attribute</i>), 217
name (<i>slixmpp.plugins.xep_0359.stanza.OriginID attribute</i>), 209	name (<i>slixmpp.plugins.xep_0394.stanza.Markup attribute</i>), 217
name (<i>slixmpp.plugins.xep_0359.stanza.StanzaID attribute</i>), 210	name (<i>slixmpp.plugins.xep_0394.stanza.Span attribute</i>), 218
name (<i>slixmpp.plugins.xep_0363.stanza.Get attribute</i>), 210	name (<i>slixmpp.plugins.xep_0403.stanza.MIXPresence attribute</i>), 218
name (<i>slixmpp.plugins.xep_0363.stanza.Header attribute</i>), 210	name (<i>slixmpp.plugins.xep_0404.stanza.Participant attribute</i>), 219
name (<i>slixmpp.plugins.xep_0363.stanza.Put attribute</i>), 210	name (<i>slixmpp.plugins.xep_0404.stanza.UserPreference attribute</i>), 219
name (<i>slixmpp.plugins.xep_0363.stanza.Request attribute</i>), 210	name (<i>slixmpp.plugins.xep_0405.stanza.ClientJoin attribute</i>), 220
name (<i>slixmpp.plugins.xep_0363.stanza.Slot attribute</i>), 211	name (<i>slixmpp.plugins.xep_0405.stanza.ClientLeave attribute</i>), 220
name (<i>slixmpp.plugins.xep_0369.stanza.Create attribute</i>), 213	name (<i>slixmpp.plugins.xep_0421.stanza.OccupantId attribute</i>), 221
name (<i>slixmpp.plugins.xep_0369.stanza.Destroy attribute</i>), 213	name (<i>slixmpp.plugins.xep_0422.stanza.ApplyTo attribute</i>), 222
name (<i>slixmpp.plugins.xep_0369.stanza.Join attribute</i>), 213	name (<i>slixmpp.plugins.xep_0422.stanza.External attribute</i>), 222
name (<i>slixmpp.plugins.xep_0369.stanza.Leave attribute</i>), 213	name (<i>slixmpp.plugins.xep_0424.stanza.Retract attribute</i>), 223
name (<i>slixmpp.plugins.xep_0369.stanza.MIX attribute</i>), 213	name (<i>slixmpp.plugins.xep_0424.stanza.Retracted attribute</i>), 223
name (<i>slixmpp.plugins.xep_0369.stanza.Participant attribute</i>), 213	name (<i>slixmpp.plugins.xep_0425.stanza.Moderate attribute</i>), 223
name (<i>slixmpp.plugins.xep_0369.stanza.Setnick attribute</i>), 214	name (<i>slixmpp.plugins.xep_0425.stanza.Moderated attribute</i>), 223
name (<i>slixmpp.plugins.xep_0369.stanza.Subscribe attribute</i>), 214	name (<i>slixmpp.plugins.xep_0428.stanza.Fallback attribute</i>), 224
name (<i>slixmpp.plugins.xep_0369.stanza.Unsubscribe attribute</i>), 214	name (<i>slixmpp.plugins.xep_0437.stanza.Activity attribute</i>), 224
name (<i>slixmpp.plugins.xep_0369.stanza.UpdateSubscription attribute</i>), 214	name (<i>slixmpp.plugins.xep_0437.stanza.RAI attribute</i>), 225
name (<i>slixmpp.plugins.xep_0377.stanza.Report attribute</i>), 215	name (<i>slixmpp.plugins.xep_0439.stanza.Action attribute</i>), 226
name (<i>slixmpp.plugins.xep_0377.stanza.Text attribute</i>), 215	name (<i>slixmpp.plugins.xep_0439.stanza.ActionSelected attribute</i>), 226
name (<i>slixmpp.plugins.xep_0380.stanza.Encryption attribute</i>), 216	name (<i>slixmpp.plugins.xep_0439.stanza.Response attribute</i>), 226
name (<i>slixmpp.plugins.xep_0394.stanza.BlockCode attribute</i>), 216	name (<i>slixmpp.plugins.xep_0444.stanza.Reaction attribute</i>), 227
name (<i>slixmpp.plugins.xep_0394.stanza.BlockQuote attribute</i>), 216	name (<i>slixmpp.plugins.xep_0444.stanza.Reactions attribute</i>), 227
name (<i>slixmpp.plugins.xep_0394.stanza.CodeType attribute</i>), 216	name (<i>slixmpp.xmlstream.handler.base.BaseHandler attribute</i>), 89
name (<i>slixmpp.plugins.xep_0394.stanza.DeletedType attribute</i>), 217	name (<i>slixmpp.xmlstream.stanzabase.ElementBase attribute</i>), 84
name (<i>slixmpp.plugins.xep_0394.stanza.EmphasisType attribute</i>), 217	namespace (<i>slixmpp.plugins.xep_0004.stanza.field.FieldOption attribute</i>), 100
name (<i>slixmpp.plugins.xep_0394.stanza.Li attribute</i>), 217	namespace (<i>slixmpp.plugins.xep_0004.stanza.field.FormField attribute</i>), 101

namespace (*slixmpp.plugins.xep_0004.stanza.form.Form* namespace (*slixmpp.plugins.xep_0047.stanza.Data attribute*), 103
 namespace (*slixmpp.plugins.xep_0009.stanza.RPC.Method* namespace (*slixmpp.plugins.xep_0047.stanza.Open attribute*), 104
 namespace (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse* namespace (*slixmpp.plugins.xep_0049.stanza.PrivateXML attribute*), 104
 namespace (*slixmpp.plugins.xep_0009.stanza.RPC.Query* namespace (*slixmpp.plugins.xep_0050.stanza.Command attribute*), 105
 namespace (*slixmpp.plugins.xep_0012.stanza.LastActivity* namespace (*slixmpp.plugins.xep_0054.stanza.Address attribute*), 105
 namespace (*slixmpp.plugins.xep_0013.stanza.Item* attribute), 106
 namespace (*slixmpp.plugins.xep_0013.stanza.Offline* attribute), 106
 namespace (*slixmpp.plugins.xep_0020.stanza.FeatureNegotiation* attribute), 107
 namespace (*slixmpp.plugins.xep_0027.stanza.Encrypted* attribute), 107
 namespace (*slixmpp.plugins.xep_0027.stanza.Signed* attribute), 108
 namespace (*slixmpp.plugins.xep_0030.stanza.info.DiscoInfo* attribute), 116
 namespace (*slixmpp.plugins.xep_0030.stanza.items.DiscoItem* attribute), 116
 namespace (*slixmpp.plugins.xep_0030.stanza.items.DiscoItem* attribute), 117
 namespace (*slixmpp.plugins.xep_0033.stanza.Address* attribute), 118
 namespace (*slixmpp.plugins.xep_0033.stanza.Addresses* attribute), 119
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCActor* attribute), 121
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCAdmin* attribute), 122
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCAdminQuery* attribute), 122
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCBase* attribute), 122
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCDecline* attribute), 123
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCHistory* attribute), 123
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCInvite* attribute), 123
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCJoin* attribute), 123
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCOwner* attribute), 124
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCStatus* attribute), 124
 namespace (*slixmpp.plugins.xep_0045.stanza.MUCUserItem* attribute), 125
 namespace (*slixmpp.plugins.xep_0047.stanza.Close* attribute), 125
 namespace (*slixmpp.plugins.xep_0054.stanza.Agent* attribute), 130
 namespace (*slixmpp.plugins.xep_0054.stanza.BirthDay* attribute), 131
 namespace (*slixmpp.plugins.xep_0054.stanza.Categories* attribute), 131
 namespace (*slixmpp.plugins.xep_0054.stanza.Classification* attribute), 132
 namespace (*slixmpp.plugins.xep_0054.stanza.Desc* attribute), 132
 namespace (*slixmpp.plugins.xep_0054.stanza.Email* attribute), 132
 namespace (*slixmpp.plugins.xep_0054.stanza.Geo* attribute), 132
 namespace (*slixmpp.plugins.xep_0054.stanza.JabberID* attribute), 133
 namespace (*slixmpp.plugins.xep_0054.stanza.Label* attribute), 133
 namespace (*slixmpp.plugins.xep_0054.stanza.Logo* attribute), 133
 namespace (*slixmpp.plugins.xep_0054.stanza.Mailer* attribute), 133
 namespace (*slixmpp.plugins.xep_0054.stanza.Name* attribute), 134
 namespace (*slixmpp.plugins.xep_0054.stanza.Nickname* attribute), 134
 namespace (*slixmpp.plugins.xep_0054.stanza.Note* attribute), 134
 namespace (*slixmpp.plugins.xep_0054.stanza.Org* attribute), 135
 namespace (*slixmpp.plugins.xep_0054.stanza.Photo* attribute), 135
 namespace (*slixmpp.plugins.xep_0054.stanza.ProdID* attribute), 135
 namespace (*slixmpp.plugins.xep_0054.stanza.Rev* attribute), 136
 namespace (*slixmpp.plugins.xep_0054.stanza.Role* attribute), 136
 namespace (*slixmpp.plugins.xep_0054.stanza.SortString* attribute), 136
 namespace (*slixmpp.plugins.xep_0054.stanza.Sound* attribute), 136

namespace (*slixmpp.plugins.xep_0054.stanza.Telephone* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventConfig*
attribute), 137 attribute), 152
namespace (*slixmpp.plugins.xep_0054.stanza.TimeZone* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDelete*
attribute), 137 attribute), 153
namespace (*slixmpp.plugins.xep_0054.stanza.Title* attribute), 138 namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventDisass*
attribute), 153
namespace (*slixmpp.plugins.xep_0054.stanza.UID* attribute), 138 namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItem*
attribute), 153
namespace (*slixmpp.plugins.xep_0054.stanza.URL* attribute), 138 namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItems*
attribute), 153
namespace (*slixmpp.plugins.xep_0054.stanza.VCardTemp* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventPurge*
attribute), 138 attribute), 153
namespace (*slixmpp.plugins.xep_0059.stanza.Set* attribute), 140 namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventRetrac*
attribute), 154
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventSubscr*
attribute), 143 attribute), 154
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.DefaultCon*
attribute), 144 attribute), 148
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Config* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffili*
attribute), 144 attribute), 149
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Create* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffili*
attribute), 144 attribute), 149
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Default* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConf*
attribute), 144 attribute), 149
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Item* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefa*
attribute), 145 attribute), 150
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Items* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDela*
attribute), 145 attribute), 150
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Options* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerPur*
attribute), 145 attribute), 150
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Publish* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedi*
attribute), 145 attribute), 150
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubsc*
attribute), 146 attribute), 151
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubsc*
attribute), 146 attribute), 151
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Retrieve* namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.PubsubOwn*
attribute), 146 attribute), 151
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe* namespace (*slixmpp.plugins.xep_0065.stanza.Socks5*
attribute), 146 attribute), 155
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions* namespace (*slixmpp.plugins.xep_0065.stanza.StreamHost*
attribute), 147 attribute), 155
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions* namespace (*slixmpp.plugins.xep_0065.stanza.StreamHostUsed*
attribute), 147 attribute), 155
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptions* namespace (*slixmpp.plugins.xep_0066.stanza.OOB*
attribute), 147 attribute), 156
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe* namespace (*slixmpp.plugins.xep_0066.stanza.OOBTransfer*
attribute), 148 attribute), 156
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_eventEvent* namespace (*slixmpp.plugins.xep_0070.stanza.Confirm*
attribute), 151 attribute), 157
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_eventEvent* namespace (*slixmpp.plugins.xep_0071.stanza.XHTML_IM*
attribute), 152 attribute), 157
namespace (*slixmpp.plugins.xep_0060.stanza.pubsub_eventEvent* namespace (*slixmpp.plugins.xep_0077.stanza.Register*
attribute), 152 attribute), 158

namespace (<i>slixmpp.plugins.xep_0077.stanza.RegisterFeature</i> attribute), 158	namespace (<i>slixmpp.plugins.xep_0184.stanza.Received</i> attribute), 178
namespace (<i>slixmpp.plugins.xep_0079.stanza.AMP</i> attribute), 159	namespace (<i>slixmpp.plugins.xep_0184.stanza.Request</i> attribute), 179
namespace (<i>slixmpp.plugins.xep_0079.stanza.AMPFeature</i> attribute), 159	namespace (<i>slixmpp.plugins.xep_0186.stanza.Invisible</i> attribute), 179
namespace (<i>slixmpp.plugins.xep_0079.stanza.FailedRule</i> attribute), 159	namespace (<i>slixmpp.plugins.xep_0186.stanza.Visible</i> attribute), 179
namespace (<i>slixmpp.plugins.xep_0079.stanza.FailedRule</i> attribute), 159	namespace (<i>slixmpp.plugins.xep_0191.stanza.BlockList</i> attribute), 180
namespace (<i>slixmpp.plugins.xep_0079.stanza.InvalidRule</i> attribute), 160	namespace (<i>slixmpp.plugins.xep_0196.stanza.UserGaming</i> attribute), 181
namespace (<i>slixmpp.plugins.xep_0079.stanza.Rule</i> attribute), 160	namespace (<i>slixmpp.plugins.xep_0198.stanza.Ack</i> attribute), 182
namespace (<i>slixmpp.plugins.xep_0079.stanza.UnsupportedAttribute</i> attribute), 160	namespace (<i>slixmpp.plugins.xep_0198.stanza.Enable</i> attribute), 182
namespace (<i>slixmpp.plugins.xep_0079.stanza.UnsupportedCondition</i> attribute), 160	namespace (<i>slixmpp.plugins.xep_0198.stanza.Enabled</i> attribute), 182
namespace (<i>slixmpp.plugins.xep_0080.stanza.Geolocation</i> attribute), 163	namespace (<i>slixmpp.plugins.xep_0198.stanza.Failed</i> attribute), 183
namespace (<i>slixmpp.plugins.xep_0084.stanza.Data</i> attribute), 164	namespace (<i>slixmpp.plugins.xep_0198.stanza.RequestAck</i> attribute), 183
namespace (<i>slixmpp.plugins.xep_0084.stanza.Info</i> attribute), 165	namespace (<i>slixmpp.plugins.xep_0198.stanza.Resume</i> attribute), 183
namespace (<i>slixmpp.plugins.xep_0084.stanza.MetaData</i> attribute), 165	namespace (<i>slixmpp.plugins.xep_0198.stanza.Resumed</i> attribute), 184
namespace (<i>slixmpp.plugins.xep_0084.stanza.Pointer</i> attribute), 165	namespace (<i>slixmpp.plugins.xep_0198.stanza.StreamManagement</i> attribute), 184
namespace (<i>slixmpp.plugins.xep_0085.stanza.ChatState</i> attribute), 166	namespace (<i>slixmpp.plugins.xep_0199.stanza.Ping</i> attribute), 185
namespace (<i>slixmpp.plugins.xep_0086.stanza.LegacyError</i> attribute), 167	namespace (<i>slixmpp.plugins.xep_0202.stanza.EntityTime</i> attribute), 186
namespace (<i>slixmpp.plugins.xep_0092.stanza.Version</i> attribute), 168	namespace (<i>slixmpp.plugins.xep_0203.stanza.Delay</i> attribute), 187
namespace (<i>slixmpp.plugins.xep_0107.stanza.UserMood</i> attribute), 169	namespace (<i>slixmpp.plugins.xep_0221.stanza.Media</i> attribute), 187
namespace (<i>slixmpp.plugins.xep_0108.stanza.UserActivity</i> attribute), 170	namespace (<i>slixmpp.plugins.xep_0221.stanza.URI</i> attribute), 188
namespace (<i>slixmpp.plugins.xep_0115.stanza.Capabilities</i> attribute), 171	namespace (<i>slixmpp.plugins.xep_0224.stanza.Attention</i> attribute), 190
namespace (<i>slixmpp.plugins.xep_0118.stanza.UserTune</i> attribute), 172	namespace (<i>slixmpp.plugins.xep_0231.stanza.BitsOfBinary</i> attribute), 191
namespace (<i>slixmpp.plugins.xep_0122.stanza.FormValidation</i> attribute), 172	namespace (<i>slixmpp.plugins.xep_0235.stanza.OAuth</i> attribute), 191
namespace (<i>slixmpp.plugins.xep_0131.stanza.Headers</i> attribute), 174	namespace (<i>slixmpp.plugins.xep_0249.stanza.Invite</i> attribute), 192
namespace (<i>slixmpp.plugins.xep_0152.stanza.Address</i> attribute), 175	namespace (<i>slixmpp.plugins.xep_0257.stanza.AppendCert</i> attribute), 193
namespace (<i>slixmpp.plugins.xep_0152.stanza.Reachability</i> attribute), 175	namespace (<i>slixmpp.plugins.xep_0257.stanza.CertItem</i> attribute), 193
namespace (<i>slixmpp.plugins.xep_0153.stanza.VCardTempUpdate</i> attribute), 175	namespace (<i>slixmpp.plugins.xep_0257.stanza.Certs</i> attribute), 194
namespace (<i>slixmpp.plugins.xep_0172.stanza.UserNick</i> attribute), 177	namespace (<i>slixmpp.plugins.xep_0257.stanza.DisableCert</i> attribute), 194

namespace (*slixmpp.plugins.xep_0257.stanza.RevokeCert* namespace (*slixmpp.plugins.xep_0333.stanza.Markable attribute*), 194
attribute), 205
namespace (*slixmpp.plugins.xep_0258.stanza.Catalog* namespace (*slixmpp.plugins.xep_0333.stanza.Received attribute*), 194
attribute), 206
namespace (*slixmpp.plugins.xep_0258.stanza.CatalogItem* namespace (*slixmpp.plugins.xep_0334.stanza.NoCopy attribute*), 195
attribute), 206
namespace (*slixmpp.plugins.xep_0258.stanza.DisplayMarking* namespace (*slixmpp.plugins.xep_0334.stanza.NoPermanentStore attribute*), 195
attribute), 206
namespace (*slixmpp.plugins.xep_0258.stanza.EquivalentLabel* namespace (*slixmpp.plugins.xep_0334.stanza.NoStore attribute*), 196
attribute), 206
namespace (*slixmpp.plugins.xep_0258.stanza.ESSLabel* namespace (*slixmpp.plugins.xep_0334.stanza.Store attribute*), 196
attribute), 206
namespace (*slixmpp.plugins.xep_0258.stanza.Label* namespace (*slixmpp.plugins.xep_0335.stanza.JSON_Container attribute*), 196
attribute), 207
namespace (*slixmpp.plugins.xep_0258.stanza.SecurityLabel* namespace (*slixmpp.plugins.xep_0352.stanza.Active attribute*), 196
attribute), 207
namespace (*slixmpp.plugins.xep_0279.stanza.IPCheck* namespace (*slixmpp.plugins.xep_0352.stanza.ClientStateIndication attribute*), 197
attribute), 208
namespace (*slixmpp.plugins.xep_0280.stanza.CarbonDisable* namespace (*slixmpp.plugins.xep_0352.stanza.Inactive attribute*), 197
attribute), 208
namespace (*slixmpp.plugins.xep_0280.stanza.CarbonEnable* namespace (*slixmpp.plugins.xep_0353.stanza.JingleMessage attribute*), 197
attribute), 208
namespace (*slixmpp.plugins.xep_0280.stanza.PrivateCarbon* namespace (*slixmpp.plugins.xep_0359.stanza.OriginID attribute*), 198
attribute), 209
namespace (*slixmpp.plugins.xep_0280.stanza.ReceivedCarbon* namespace (*slixmpp.plugins.xep_0359.stanza.StanzaID attribute*), 198
attribute), 210
namespace (*slixmpp.plugins.xep_0280.stanza.SentCarbon* namespace (*slixmpp.plugins.xep_0363.stanza.Get attribute*), 198
attribute), 210
namespace (*slixmpp.plugins.xep_0297.stanza.Forwarded* namespace (*slixmpp.plugins.xep_0363.stanza.Header attribute*), 199
attribute), 210
namespace (*slixmpp.plugins.xep_0300.stanza.Hash* attribute), 199
namespace (*slixmpp.plugins.xep_0308.stanza.Replace* attribute), 200
namespace (*slixmpp.plugins.xep_0313.stanza.Fin* attribute), 201
namespace (*slixmpp.plugins.xep_0313.stanza.MAM* attribute), 201
namespace (*slixmpp.plugins.xep_0313.stanza.Preferences* attribute), 201
namespace (*slixmpp.plugins.xep_0313.stanza.Result* attribute), 202
namespace (*slixmpp.plugins.xep_0319.stanza.Idle* attribute), 202
namespace (*slixmpp.plugins.xep_0332.stanza.data.HTTPData* attribute), 203
namespace (*slixmpp.plugins.xep_0332.stanza.request.HTTPRequest* attribute), 204
namespace (*slixmpp.plugins.xep_0332.stanza.response.HTTPResponse* attribute), 204
namespace (*slixmpp.plugins.xep_0333.stanza.Acknowledged* attribute), 205
namespace (*slixmpp.plugins.xep_0333.stanza.Displayed* attribute), 205
namespace (*slixmpp.plugins.xep_0333.stanza.Markable* attribute), 205
namespace (*slixmpp.plugins.xep_0333.stanza.Received* attribute), 206
namespace (*slixmpp.plugins.xep_0334.stanza.NoCopy* attribute), 206
namespace (*slixmpp.plugins.xep_0334.stanza.NoPermanentStore* attribute), 206
namespace (*slixmpp.plugins.xep_0334.stanza.Store* attribute), 206
namespace (*slixmpp.plugins.xep_0335.stanza.JSON_Container* attribute), 207
namespace (*slixmpp.plugins.xep_0352.stanza.Active* attribute), 207
namespace (*slixmpp.plugins.xep_0352.stanza.ClientStateIndication* attribute), 208
namespace (*slixmpp.plugins.xep_0352.stanza.Inactive* attribute), 208
namespace (*slixmpp.plugins.xep_0353.stanza.JingleMessage* attribute), 208
namespace (*slixmpp.plugins.xep_0359.stanza.OriginID* attribute), 209
namespace (*slixmpp.plugins.xep_0359.stanza.StanzaID* attribute), 210
namespace (*slixmpp.plugins.xep_0363.stanza.Get* attribute), 210
namespace (*slixmpp.plugins.xep_0363.stanza.Header* attribute), 210
namespace (*slixmpp.plugins.xep_0363.stanza.Put* attribute), 210
namespace (*slixmpp.plugins.xep_0363.stanza.Request* attribute), 211
namespace (*slixmpp.plugins.xep_0363.stanza.Slot* attribute), 211
namespace (*slixmpp.plugins.xep_0369.stanza.Create* attribute), 213
namespace (*slixmpp.plugins.xep_0369.stanza.Destroy* attribute), 213
namespace (*slixmpp.plugins.xep_0369.stanza.Join* attribute), 213
namespace (*slixmpp.plugins.xep_0369.stanza.Leave* attribute), 213
namespace (*slixmpp.plugins.xep_0369.stanza.MIX* attribute), 213
namespace (*slixmpp.plugins.xep_0369.stanza.Participant* attribute), 214
namespace (*slixmpp.plugins.xep_0369.stanza.Setnick* attribute), 214
namespace (*slixmpp.plugins.xep_0369.stanza.Subscribe* attribute), 214
namespace (*slixmpp.plugins.xep_0369.stanza.Unsubscribe* attribute), 214

namespace (*slixmpp.plugins.xep_0369.stanza.UpdateSubscription*)
attribute), 214

namespace (*slixmpp.plugins.xep_0377.stanza.Report*)
attribute), 215

namespace (*slixmpp.plugins.xep_0377.stanza.Text*)
attribute), 215

namespace (*slixmpp.plugins.xep_0380.stanza.Encryption*)
attribute), 216

namespace (*slixmpp.plugins.xep_0394.stanza.Liaison*)
attribute), 217

namespace (*slixmpp.plugins.xep_0394.stanza.Markup*)
attribute), 217

namespace (*slixmpp.plugins.xep_0403.stanza.MIXPresence*)
attribute), 218

namespace (*slixmpp.plugins.xep_0404.stanza.Participant*)
NoCopy (*class in slixmpp.plugins.xep_0334.stanza*), 206
attribute), 219

namespace (*slixmpp.plugins.xep_0404.stanza.UserPreference*)
attribute), 219

namespace (*slixmpp.plugins.xep_0405.stanza.ClientJoin*)
attribute), 220

namespace (*slixmpp.plugins.xep_0405.stanza.ClientLeave*)
Note (*class in slixmpp.plugins.xep_0054.stanza*), 134
attribute), 220

namespace (*slixmpp.plugins.xep_0421.stanza.OccupantId*)
attribute), 221

namespace (*slixmpp.plugins.xep_0422.stanza.ApplyTo*)
attribute), 222

namespace (*slixmpp.plugins.xep_0422.stanza.External*)
attribute), 222

namespace (*slixmpp.plugins.xep_0424.stanza.Retract*)
attribute), 223

namespace (*slixmpp.plugins.xep_0424.stanza.Retracted*)
attribute), 223

namespace (*slixmpp.plugins.xep_0425.stanza.Moderate*)
attribute), 223

namespace (*slixmpp.plugins.xep_0425.stanza.Moderated*)
attribute), 223

namespace (*slixmpp.plugins.xep_0428.stanza.Fallback*)
attribute), 224

namespace (*slixmpp.plugins.xep_0437.stanza.Activity*)
attribute), 224

namespace (*slixmpp.plugins.xep_0437.stanza.RAI*)
attribute), 225

namespace (*slixmpp.plugins.xep_0439.stanza.Action*)
attribute), 226

namespace (*slixmpp.plugins.xep_0439.stanza.ActionSelected*)
attribute), 226

namespace (*slixmpp.plugins.xep_0439.stanza.Response*)
attribute), 226

namespace (*slixmpp.plugins.xep_0444.stanza.Reaction*)
attribute), 227

namespace (*slixmpp.plugins.xep_0444.stanza.Reactions*)
attribute), 227

namespace (*slixmpp.xmlstream.stanzabase.ElementBase*)
attribute), 84

namespace (*slixmpp.xmlstream.stanzabase.StanzaBase*)
attribute), 88

namespace_map (*slixmpp.xmlstream.xmlstream.XMLStream*)
attribute), 97

new_id () (*slixmpp.xmlstream.xmlstream.XMLStream*)
method), 97

new_session () (*slixmpp.plugins.xep_0050.XEP_0050*)
method), 127

next () (*slixmpp.xmlstream.stanzabase.ElementBase*)
method), 85

next_actions (*slixmpp.plugins.xep_0050.stanza.Command*)
attribute), 129

nickname (*class in slixmpp.plugins.xep_0054.stanza*), 134

NoCopy (*class in slixmpp.plugins.xep_0334.stanza*), 206

NoPermanentStore (*class* in
slixmpp.plugins.xep_0334.stanza), 206

normal () (*slixmpp.stanza.Message*)
method), 229

NoStore (*class in slixmpp.plugins.xep_0334.stanza*), 206

Note (*class in slixmpp.plugins.xep_0054.stanza*), 134

OAuth (*class in slixmpp.plugins.xep_0235.stanza*), 191

OccupantId (*class* in
slixmpp.plugins.xep_0421.stanza), 221

Offline (*class in slixmpp.plugins.xep_0013.stanza*), 106

OOB (*class in slixmpp.plugins.xep_0066.stanza*), 156

OOBTransfer (*class* in
slixmpp.plugins.xep_0066.stanza), 156

Open (*class in slixmpp.plugins.xep_0047.stanza*), 125

option_types (*slixmpp.plugins.xep_0004.stanza.field.FormField*)
attribute), 101

OptionalSetting (*class* in
slixmpp.plugins.xep_0060.stanza.base), 143

Options (*class in slixmpp.plugins.xep_0060.stanza.pubsub*), 145

Org (*class in slixmpp.plugins.xep_0054.stanza*), 135

OriginID (*class in slixmpp.plugins.xep_0359.stanza*), 209

overrides (*slixmpp.plugins.xep_0086.stanza.LegacyError*)
attribute), 168

overrides (*slixmpp.xmlstream.stanzabase.ElementBase*)
attribute), 85

OwnerAffiliation (*class* in
slixmpp.plugins.xep_0060.stanza.pubsub_owner), 149

OwnerAffiliations (*class* in
slixmpp.plugins.xep_0060.stanza.pubsub_owner), 149

OwnerConfigure (*class* in
slixmpp.plugins.xep_0060.stanza.pubsub_owner), 149

```

OwnerDefault          (class      in plugin_attrib(slixmpp.plugins.xep_0020.stanza.FeatureNegotiation
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),       attribute), 107
    149                           plugin_attrib(slixmpp.plugins.xep_0027.stanza.Encrypted

OwnerDelete          (class      in      attribute), 107
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),plugin_attrib(slixmpp.plugins.xep_0027.stanza.Signed
    150                           attribute), 108

OwnerPurge           (class      in plugin_attrib(slixmpp.plugins.xep_0030.stanza.info.DiscoInfo
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),       attribute), 116
    150                           plugin_attrib(slixmpp.plugins.xep_0030.stanza.items.DiscoItem

OwnerRedirect         (class      in      attribute), 116
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),plugin_attrib(slixmpp.plugins.xep_0030.stanza.items.DiscoItems
    150                           attribute), 117

OwnerSubscription     (class      in plugin_attrib(slixmpp.plugins.xep_0033.stanza.Address
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),       attribute), 118
    150                           plugin_attrib(slixmpp.plugins.xep_0033.stanza.Addresses

OwnerSubscriptions   (class      in      attribute), 119
    slixmpp.plugins.xep_0060.stanza.pubsub_owner),plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCActor
    151                           attribute), 122
                                            plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCAdminItem
                                            attribute), 122

P

parent  (slixmpp.xmlstream.stanzabase.ElementBase      attribute), 85
Participant        (class      in plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCAdminQuery
    slixmpp.plugins.xep_0369.stanza), 213
Participant        (class      in plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCDecline
    slixmpp.plugins.xep_0404.stanza), 219
Paused            (class in slixmpp.plugins.xep_0085.stanza), 166
Photo             (class in slixmpp.plugins.xep_0054.stanza), 135
pick_dns_answer() plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCInvite
                           attribute), 123
                           plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCJoin
                           attribute), 123
Ping              (class in slixmpp.plugins.xep_0199.stanza), 185
ping()            (slixmpp.plugins.xep_0199.XEP_0199 method), 185
plugin            (slixmpp.basexmpp.BaseXMPP attribute), 72
plugin_attrib     (slixmpp.plugins.xep_0004.stanza.field.FieldOption attribute), 124
                           attribute), 100
                           plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCStatus
                           attribute), 124
plugin_attrib     (slixmpp.plugins.xep_0004.stanza.field.FormField attribute), 124
                           attribute), 101
                           plugin_attrib(slixmpp.plugins.xep_0045.stanza.MUCUserItem
                           attribute), 125
plugin_attrib     (slixmpp.plugins.xep_0004.stanza.form.Form attribute), 125
                           attribute), 103
                           plugin_attrib(slixmpp.plugins.xep_0047.stanza.Close
                           attribute), 125
                           plugin_attrib(slixmpp.plugins.xep_0047.stanza.Data
                           attribute), 104
                           plugin_attrib(slixmpp.plugins.xep_0047.stanza.Open
                           attribute), 125
                           plugin_attrib(slixmpp.plugins.xep_0047.stanza.PrivateXML
                           attribute), 105
                           plugin_attrib(slixmpp.plugins.xep_0050.stanza.Command
                           attribute), 126
                           plugin_attrib(slixmpp.plugins.xep_0050.stanza.LastActivity
                           attribute), 105
                           plugin_attrib(slixmpp.plugins.xep_0054.stanza.Address
                           attribute), 106
                           plugin_attrib(slixmpp.plugins.xep_0054.stanza.Agent
                           attribute), 130
                           plugin_attrib(slixmpp.plugins.xep_0054.stanza.Agent
                           attribute), 130

```

plugin_attrib(*slixmpp.plugins.xep_0054.stanza.BinValue*, 131) plugin_attrib(*slixmpp.plugins.xep_0059.stanza.SetAttribute*), 140
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Birthdate*, 131) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.AffiliationAttribute*), 143
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Category*, 131) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.AffiliationsAttribute*), 144
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Classification*, 132) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.ConfigureAttribute*), 144
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Description*, 132) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.CreateAttribute*), 144
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Email*, 132) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.DefaultAttribute*), 144
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Geo*, 132) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.ItemAttribute*), 145
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.JabberID*, 133) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.ItemsAttribute*), 145
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Label*, 133) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.OptionsAttribute*), 145
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Logo*, 133) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.PublishAttribute*), 145
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Mailbox*, 134) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptionsAttribute*), 146
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Name*, 134) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.PubsubAttribute*), 146
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Nickname*, 134) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.RetractAttribute*), 146
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Note*, 135) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeAttribute*), 147
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Org*, 135) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.SubscribeOptionsAttribute*), 147
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Photo*, 135) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.SubscriptionAttribute*), 147
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.ProductID*, 135) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.SubscriptionsAttribute*), 147
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Rev*, 136) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub.UnsubscribeAttribute*), 148
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Role*, 136) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubAttribute*), 148
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.SortString*, 136) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventAttribute*), 151
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Sound*, 136) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventAAttribute*), 152
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Telephone*, 137) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCAttribute*), 152
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.TimeZone*, 137) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCAttribute*), 152
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.Title*, 138) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventLAttribute*), 153
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.UID*, 138) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventLAttribute*), 153
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.URL*, 138) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventIAttribute*), 153
 plugin_attrib(*slixmpp.plugins.xep_0054.stanza.VCardTemp*, 138) plugin_attrib(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventIAttribute*), 153

plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\event\EventPublish*, *slixmpp.plugins.xep_0084.stanza.Data attribute*), 153
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\event\EventRetract*, *slixmpp.plugins.xep_0084.stanza.Info attribute*), 154
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\event\EventSends*, *slixmpp.plugins.xep_0084.stanza.MetaData attribute*), 154
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\DefaultConfig*, *slixmpp.plugins.xep_0084.stanza.Pointer attribute*), 149
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerConfig*, *slixmpp.plugins.xep_0085.stanza.ChatState attribute*), 149
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerDelete*, *slixmpp.plugins.xep_0086.stanza.LegacyError attribute*), 150
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerPing*, *slixmpp.plugins.xep_0092.stanza.Version attribute*), 150
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerRedist*, *slixmpp.plugins.xep_0107.stanza.UserMood attribute*), 150
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerSend*, *slixmpp.plugins.xep_0108.stanza.UserActivity attribute*), 151
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PointerSend*, *slixmpp.plugins.xep_0115.stanza.Capabilities attribute*), 151
plugin_attrib (*slixmpp.plugins.xep_0060.stanza.pubsub\log\PathSubj*, *slixmpp.plugins.xep_0118.stanza.UserTune attribute*), 151
plugin_attrib (*slixmpp.plugins.xep_0065.stanza.Socks5Plugin_attrib*, *slixmpp.plugins.xep_0122.stanza.FormValidation attribute*), 155
plugin_attrib (*slixmpp.plugins.xep_0065.stanza.StreamHostIn_attrib*, *slixmpp.plugins.xep_0131.stanza.Headers attribute*), 155
plugin_attrib (*slixmpp.plugins.xep_0065.stanza.StreamHostUsed_attrib*, *slixmpp.plugins.xep_0152.stanza.Address attribute*), 155
plugin_attrib (*slixmpp.plugins.xep_0066.stanza.OOBPlugin_attrib*, *slixmpp.plugins.xep_0152.stanza.Reachability attribute*), 156
plugin_attrib (*slixmpp.plugins.xep_0066.stanza.OOBTransfer_attrib*, *slixmpp.plugins.xep_0153.stanza.VCardTempUpdate attribute*), 156
plugin_attrib (*slixmpp.plugins.xep_0070.stanza.ConfigPlugin_attrib*, *slixmpp.plugins.xep_0172.stanza.UserNick attribute*), 157
plugin_attrib (*slixmpp.plugins.xep_0071.stanza.XHTMLIMAttrib*, *slixmpp.plugins.xep_0184.stanza.Received attribute*), 157
plugin_attrib (*slixmpp.plugins.xep_0077.stanza.RegisterPlugin_attrib*, *slixmpp.plugins.xep_0184.stanza.Request attribute*), 158
plugin_attrib (*slixmpp.plugins.xep_0077.stanza.RegisterFeatureAttrib*, *slixmpp.plugins.xep_0186.stanza.Invisible attribute*), 158
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.AMPPlugin_attrib*, *slixmpp.plugins.xep_0186.stanza.Visible attribute*), 159
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.FailedRulesAttrib*, *slixmpp.plugins.xep_0191.stanza.Block attribute*), 159
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.InvalidRuleAttrib*, *slixmpp.plugins.xep_0191.stanza.BlockList attribute*), 160
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.Rule plugin_attrib*, *slixmpp.plugins.xep_0191.stanza.Unblock attribute*), 160
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.UnsupportedActionAttrib*, *slixmpp.plugins.xep_0196.stanza.UserGaming attribute*), 160
plugin_attrib (*slixmpp.plugins.xep_0079.stanza.UnsupportedConditionAttrib*, *slixmpp.plugins.xep_0198.stanza.StreamManagement attribute*), 160
plugin_attrib (*slixmpp.plugins.xep_0080.stanza.GeolocationPlugin_attrib*, *slixmpp.plugins.xep_0199.stanza.Ping attribute*), 163

plugin_attrib (*slixmpp.plugins.xep_0202.stanza.EntityTime*), 186
 plugin_attrib (*slixmpp.plugins.xep_0203.stanza.Delay*), 187
 plugin_attrib (*slixmpp.plugins.xep_0221.stanza.Media*), 188
 plugin_attrib (*slixmpp.plugins.xep_0221.stanza.URI*), 188
 plugin_attrib (*slixmpp.plugins.xep_0224.stanza.Attention*), 190
 plugin_attrib (*slixmpp.plugins.xep_0231.stanza.BitsOfBinary*), 191
 plugin_attrib (*slixmpp.plugins.xep_0235.stanza.OAuth*), 191
 plugin_attrib (*slixmpp.plugins.xep_0249.stanza.Invite*), 192
 plugin_attrib (*slixmpp.plugins.xep_0257.stanza.Append*), 193
 plugin_attrib (*slixmpp.plugins.xep_0257.stanza.Cert*), 193
 plugin_attrib (*slixmpp.plugins.xep_0257.stanza.Certs*), 194
 plugin_attrib (*slixmpp.plugins.xep_0257.stanza.Disable*), 194
 plugin_attrib (*slixmpp.plugins.xep_0257.stanza.Revoke*), 194
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.Catalog*), 195
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.CatalogItem*), 195
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.DisplayMarking*), 195
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.EquivalentLabel*), 196
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.ESSLabel*), 196
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.Label*), 196
 plugin_attrib (*slixmpp.plugins.xep_0258.stanza.SecurityLabel*), 196
 plugin_attrib (*slixmpp.plugins.xep_0279.stanza.IPCheck*), 197
 plugin_attrib (*slixmpp.plugins.xep_0280.stanza.CarbonDisable*), 197
 plugin_attrib (*slixmpp.plugins.xep_0280.stanza.CarbonEnable*), 198
 plugin_attrib (*slixmpp.plugins.xep_0280.stanza.PrivateCarbon*), 198
 plugin_attrib (*slixmpp.plugins.xep_0280.stanza.ReceiveCarbon*), 198
 plugin_attrib (*slixmpp.plugins.xep_0280.stanza.SentCarbon*), 198
 plugin_attrib (*slixmpp.plugins.xep_0297.stanza.Forwarded*), 199
 plugin_attrib (*slixmpp.plugins.xep_0300.stanza.Hash*), 199
 plugin_attrib (*slixmpp.plugins.xep_0308.stanza.Replace*), 200
 plugin_attrib (*slixmpp.plugins.xep_0313.stanza.Finish*), 201
 plugin_attrib (*slixmpp.plugins.xep_0313.stanza.MAM*), 201
 plugin_attrib (*slixmpp.plugins.xep_0313.stanza.Preferences*), 201
 plugin_attrib (*slixmpp.plugins.xep_0313.stanza.Result*), 202
 plugin_attrib (*slixmpp.plugins.xep_0319.stanza.Idle*), 202
 plugin_attrib (*slixmpp.plugins.xep_0332.stanza.data.HTTPData*), 203
 plugin_attrib (*slixmpp.plugins.xep_0332.stanza.request.HTTPRequest*), 204
 plugin_attrib (*slixmpp.plugins.xep_0332.stanza.response.HTTPResponse*), 204
 plugin_attrib (*slixmpp.plugins.xep_0333.stanza.Acknowledged*), 205
 plugin_attrib (*slixmpp.plugins.xep_0333.stanza.Displayed*), 205
 plugin_attrib (*slixmpp.plugins.xep_0333.stanza.Markable*), 205
 plugin_attrib (*slixmpp.plugins.xep_0333.stanza.Received*), 206
 plugin_attrib (*slixmpp.plugins.xep_0334.stanza.NoCopy*), 206
 plugin_attrib (*slixmpp.plugins.xep_0334.stanza.NoPermanentStore*), 206
 plugin_attrib (*slixmpp.plugins.xep_0334.stanza.NoStore*), 206
 plugin_attrib (*slixmpp.plugins.xep_0334.stanza.Store*), 206
 plugin_attrib (*slixmpp.plugins.xep_0335.stanza.JSONContainer*), 207
 plugin_attrib (*slixmpp.plugins.xep_0352.stanza.Active*), 207
 plugin_attrib (*slixmpp.plugins.xep_0352.stanza.ClientStateIndication*), 208
 plugin_attrib (*slixmpp.plugins.xep_0352.stanza.Inactive*), 208
 plugin_attrib (*slixmpp.plugins.xep_0353.stanza.Accept*), 208
 plugin_attrib (*slixmpp.plugins.xep_0353.stanza.Proceed*), 209
 plugin_attrib (*slixmpp.plugins.xep_0353.stanza.Propose*), 209
 plugin_attrib (*slixmpp.plugins.xep_0353.stanza.Reject*), 209
 plugin_attrib (*slixmpp.plugins.xep_0353.stanza.Retract*), 209

plugin_attrib (*slixmpp.plugins.xep_0359.stanza.OriginID*, 209)
Digin_attrib (*slixmpp.plugins.xep_0394.stanza.Markup*
attribute), 217
plugin_attrib (*slixmpp.plugins.xep_0359.stanza.StanzaID*, 210)
Digin_attrib (*slixmpp.plugins.xep_0394.stanza.Span*
attribute), 218
plugin_attrib (*slixmpp.plugins.xep_0363.stanza.Get*, 210)
plugin_attrib (*slixmpp.plugins.xep_0403.stanza.MIXPresence*
attribute), 218
plugin_attrib (*slixmpp.plugins.xep_0363.stanza.Header*, 210)
plugin_attrib (*slixmpp.plugins.xep_0404.stanza.Participant*
attribute), 219
plugin_attrib (*slixmpp.plugins.xep_0363.stanza.Put*, 210)
plugin_attrib (*slixmpp.plugins.xep_0404.stanza.UserPreference*
attribute), 219
plugin_attrib (*slixmpp.plugins.xep_0363.stanza.Request*, 211)
plugin_attrib (*slixmpp.plugins.xep_0405.stanza.ClientJoin*
attribute), 220
plugin_attrib (*slixmpp.plugins.xep_0363.stanza.Slot*, 211)
plugin_attrib (*slixmpp.plugins.xep_0405.stanza.ClientLeave*
attribute), 220
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Create*, 213)
plugin_attrib (*slixmpp.plugins.xep_0421.stanza.OccupantId*
attribute), 221
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Destroy*, 213)
plugin_attrib (*slixmpp.plugins.xep_0422.stanza.ApplyTo*
attribute), 222
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Join*, 213)
plugin_attrib (*slixmpp.plugins.xep_0422.stanza.External*
attribute), 222
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Leave*, 213)
plugin_attrib (*slixmpp.plugins.xep_0424.stanza.Retract*
attribute), 223
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.MIX*, 213)
plugin_attrib (*slixmpp.plugins.xep_0424.stanza.Retracted*
attribute), 223
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Participant*, 214)
plugin_attrib (*slixmpp.plugins.xep_0425.stanza.Moderate*
attribute), 223
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Setnick*, 214)
plugin_attrib (*slixmpp.plugins.xep_0425.stanza.Moderated*
attribute), 224
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Subscribe*, 214)
begin_attrib (*slixmpp.plugins.xep_0428.stanza.Fallback*
attribute), 224
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.Unsubscribe*, 214)
begin_attrib (*slixmpp.plugins.xep_0437.stanza.Activity*
attribute), 224
plugin_attrib (*slixmpp.plugins.xep_0369.stanza.UpdateSubscription*, 214)
attribrib (*slixmpp.plugins.xep_0437.stanza.RAI*
attribute), 225
plugin_attrib (*slixmpp.plugins.xep_0377.stanza.Report*, 215)
plugin_attrib (*slixmpp.plugins.xep_0439.stanza.Action*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0377.stanza.Text*, 215)
plugin_attrib (*slixmpp.plugins.xep_0439.stanza.ActionSelected*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0380.stanza.Encryption*, 216)
begin_attrib (*slixmpp.plugins.xep_0439.stanza.Response*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.BlockCode*, 216)
begin_attrib (*slixmpp.plugins.xep_0444.stanza.Reactions*
attribute), 227
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.BlockQuot*, 216)
begin_attrib (*slixmpp.xmlstream.stanzabase.ElementBase*
attribute), 85
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.CodeType*, 216)
begin_attrib (*slixmpp.plugins.xep_0004.stanza.field.FormField*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.DeletedType*, 217)
begin_attrib (*slixmpp.plugins.xep_0004.stanza.field.FormField*
attribute), 101
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.EmphasisType*, 217)
begin_attrib (*slixmpp.plugins.xep_0009.stanza.RPC.MethodCall*
attribute), 104
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.List*, 217)
begin_attrib (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse*
attribute), 104
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.Li*, 217)
begin_attrib_map (*slixmpp.plugins.xep_0004.stanza.field.FormField*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.Li*, 217)
begin_attrib_map (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse*
attribute), 226
plugin_attrib (*slixmpp.plugins.xep_0394.stanza.List*, 217)
begin_attrib_map (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse*
attribute), 226

(*slixmpp.plugins.xep_0009.stanza.RPC.RPCQuery attribute*), 105

plugin_attrib_map
(*slixmpp.plugins.xep_0013.stanza.Offline attribute*), 106

plugin_attrib_map
(*slixmpp.plugins.xep_0030.stanza.items.DiscoItems attribute*), 117

plugin_attrib_map
(*slixmpp.plugins.xep_0033.stanza.Addresses attribute*), 119

plugin_attrib_map
(*slixmpp.plugins.xep_0054.stanza.Agent attribute*), 130

plugin_attrib_map
(*slixmpp.plugins.xep_0054.stanza.Logo attribute*), 133

plugin_attrib_map
(*slixmpp.plugins.xep_0054.stanza.Photo attribute*), 135

plugin_attrib_map
(*slixmpp.plugins.xep_0054.stanza.Sound attribute*), 136

plugin_attrib_map
(*slixmpp.plugins.xep_0054.stanza.VCardTemp attribute*), 138

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations attribute*), 144

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Configure attribute*), 144

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Items attribute*), 145

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Publish attribute*), 145

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub attribute*), 146

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Retract attribute*), 146

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute*), 147

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Subscription attribute*), 147

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub.Subscriptions attribute*), 147

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCodes attribute*), 148

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute*), 152

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventCollection attribute*), 152

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventConfiguration attribute*), 152

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_event.EventItems attribute*), 153

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.DefaultConfig attribute*), 149

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerAffiliation attribute*), 149

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerConfiguration attribute*), 149

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDefault attribute*), 150

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerDelete attribute*), 150

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscribe attribute*), 151

plugin_attrib_map
(*slixmpp.plugins.xep_0060.stanza.pubsub_owner.PubsubOwner attribute*), 151

plugin_attrib_map
(*slixmpp.plugins.xep_0065.stanza.Socks5 attribute*), 155

plugin_attrib_map
(*slixmpp.plugins.xep_0079.stanza.AMP attribute*), 159

plugin_attrib_map
(*slixmpp.plugins.xep_0079.stanza.FailedRules attribute*), 160

plugin_attrib_map
(*slixmpp.plugins.xep_0079.stanza.InvalidRules attribute*), 160

plugin_attrib_map
(*slixmpp.plugins.xep_0079.stanza.UnsupportedActions attribute*), 160

plugin_attrib_map
(*slixmpp.plugins.xep_0079.stanza.UnsupportedConditions attribute*), 160

(*slixmpp.plugins.xep_0084.stanza.MetaData attribute*), 165
plugin_attrib_map
(*slixmpp.plugins.xep_0122.stanza.FormValidation attribute*), 173
plugin_attrib_map
(*slixmpp.plugins.xep_0152.stanza.Reachability attribute*), 175
plugin_attrib_map
(*slixmpp.plugins.xep_0221.stanza.Media attribute*), 188
plugin_attrib_map
(*slixmpp.plugins.xep_0257.stanza.Certs attribute*), 194
plugin_attrib_map
(*slixmpp.plugins.xep_0258.stanza.Catalog attribute*), 195
plugin_attrib_map
(*slixmpp.plugins.xep_0258.stanza.CatalogItem attribute*), 195
plugin_attrib_map
(*slixmpp.plugins.xep_0258.stanza.EquivalentLabel attribute*), 196
plugin_attrib_map
(*slixmpp.plugins.xep_0258.stanza.Label attribute*), 196
plugin_attrib_map
(*slixmpp.plugins.xep_0258.stanza.SecurityLabel attribute*), 196
plugin_attrib_map
(*slixmpp.plugins.xep_0394.stanza.List attribute*), 217
plugin_attrib_map
(*slixmpp.plugins.xep_0394.stanza.Markup attribute*), 217
plugin_attrib_map
(*slixmpp.plugins.xep_0394.stanza.Span attribute*), 218
plugin_attrib_map
(*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 85
plugin_config (*slixmpp.basexmpp.BaseXMPP attribute*), 72
plugin_iterables (*slixmpp.plugins.xep_0013.stanza.Offline attribute*), 106
plugin_iterables (*slixmpp.plugins.xep_0030.stanza.ItemsDiscoverable attribute*), 117
plugin_iterables (*slixmpp.plugins.xep_0033.stanza.Addresses attribute*), 119
plugin_iterables (*slixmpp.plugins.xep_0054.stanza.Agent attribute*), 130
plugin_iterables (*slixmpp.plugins.xep_0054.stanza.Logon attribute*), 133
plugin_iterables (*slixmpp.plugins.xep_0054.stanza.Roster attribute*), 135
plugin_iterables (*slixmpp.plugins.xep_0054.stanza.Sound attribute*), 137
plugin_iterables (*slixmpp.plugins.xep_0054.stanza.VCardTemp attribute*), 138
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation attribute*), 144
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Configuration attribute*), 144
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Items attribute*), 145
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Publish attribute*), 145
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub attribute*), 146
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Retract attribute*), 146
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscription attribute*), 147
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscription attribute*), 147
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute*), 147
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute*), 147
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute*), 147
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute*), 152
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute*), 152
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.DiscoItems attribute*), 149
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owned attribute*), 149
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute*), 150
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute*), 150
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute*), 151
plugin_iterables (*slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute*), 151
plugin_iterables (*slixmpp.plugins.xep_0065.stanza.Socks5 attribute*), 155
plugin_iterables (*slixmpp.plugins.xep_0079.stanza.AMP attribute*), 159
plugin_iterables (*slixmpp.plugins.xep_0079.stanza.FailedRules attribute*), 160
plugin_iterables (*slixmpp.plugins.xep_0079.stanza.InvalidRules attribute*), 160
plugin_iterables (*slixmpp.plugins.xep_0079.stanza.UnsupportedActions attribute*), 160

attribute), 160
`plugin_iterables(slixmpp.plugins.xep_0079.stanza.UnsupportedContent)`
attribute), 160
`plugin_iterables(slixmpp.plugins.xep_0084.stanza.MetaData)`
attribute), 165
`plugin_iterables(slixmpp.plugins.xep_0152.stanza.Readability)`
attribute), 175
`plugin_iterables(slixmpp.plugins.xep_0221.stanza.Media)`
attribute), 188
`plugin_iterables(slixmpp.plugins.xep_0257.stanza.Certs)`
attribute), 194
`plugin_iterables(slixmpp.plugins.xep_0258.stanza.CatalogItem)`
attribute), 195
`plugin_iterables(slixmpp.plugins.xep_0258.stanza.EquivalentId)`
attribute), 196
`plugin_iterables(slixmpp.plugins.xep_0258.stanza.Label)`
attribute), 196
`plugin_iterables(slixmpp.plugins.xep_0258.stanza.SecurityLabel)`
attribute), 196
`plugin_iterables(slixmpp.plugins.xep_0394.stanza.List)`
attribute), 217
`plugin_iterables(slixmpp.plugins.xep_0394.stanza.Markup)`
attribute), 217
`plugin_iterables(slixmpp.plugins.xep_0394.stanza.Span)`
attribute), 218
`plugin_iterables(slixmpp.xmlstream.stanzabase.ElementBase)`
attribute), 85
`plugin_multi_attrib`
(slixmpp.plugins.xep_0004.stanza.field.FieldOption)
attribute), 100
`plugin_multi_attrib`
(slixmpp.plugins.xep_0004.stanza.field.FormField)
attribute), 101
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Address)
attribute), 130
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Agent)
attribute), 130
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Birthday)
attribute), 131
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Categories)
attribute), 131
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Classification)
attribute), 132
`plugin_multi_attrib`
(slixmpp.plugins.xep_0054.stanza.Desc)
attribute), 132
`plugin_multi_attrib`

(slixmpp.plugins.xep_0054.stanza.Email)
(slixmpp.plugins.xep_0054.stanza.JabberID)
(slixmpp.plugins.xep_0054.stanza.Label)
(slixmpp.plugins.xep_0054.stanza.Logo)
(slixmpp.plugins.xep_0054.stanza.Org)
(slixmpp.plugins.xep_0054.stanza.Role)
(slixmpp.plugins.xep_0054.stanza.SortString)
(slixmpp.plugins.xep_0054.stanza.Sound)
(slixmpp.plugins.xep_0054.stanza.Telephone)
(slixmpp.plugins.xep_0054.stanza.TimeZone)
(slixmpp.plugins.xep_0054.stanza.Title)

(<i>slixmpp.plugins.xep_0054.stanza.UID</i> attribute), 138	at-	<i>attribute</i>), 106
plugin_multi_attrib (<i>slixmpp.plugins.xep_0054.stanza.URL</i> attribute), 138	at-	plugin_overrides (<i>slixmpp.plugins.xep_0030.stanza.items.DiscoItem</i> attribute), 117
plugin_multi_attrib (<i>slixmpp.plugins.xep_0065.stanza.StreamHost</i> attribute), 155	at-	plugin_overrides (<i>slixmpp.plugins.xep_0033.stanza.Addresses</i> attribute), 119
plugin_multi_attrib (<i>slixmpp.plugins.xep_0079.stanza.Rule</i> attribute), 160	at-	plugin_overrides (<i>slixmpp.plugins.xep_0054.stanza.Agent</i> attribute), 130
plugin_multi_attrib (<i>slixmpp.plugins.xep_0084.stanza.Info</i> attribute), 165	at-	plugin_overrides (<i>slixmpp.plugins.xep_0054.stanza.Logo</i> attribute), 133
plugin_multi_attrib (<i>slixmpp.plugins.xep_0084.stanza.Pointer</i> attribute), 165	at-	plugin_overrides (<i>slixmpp.plugins.xep_0054.stanza.Photo</i> attribute), 135
plugin_multi_attrib (<i>slixmpp.plugins.xep_0152.stanza.Address</i> attribute), 175	at-	plugin_overrides (<i>slixmpp.plugins.xep_0054.stanza.Sound</i> attribute), 137
plugin_multi_attrib (<i>slixmpp.plugins.xep_0221.stanza.URI</i> attribute), 188	at-	plugin_overrides (<i>slixmpp.plugins.xep_0054.stanza.VCardTemp</i> attribute), 138
plugin_multi_attrib (<i>slixmpp.plugins.xep_0257.stanza.CertItem</i> attribute), 193	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation</i> attribute), 144
plugin_multi_attrib (<i>slixmpp.plugins.xep_0258.stanza.CatalogItem</i> attribute), 195	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Configuration</i> attribute), 144
plugin_multi_attrib (<i>slixmpp.plugins.xep_0258.stanza.EquivalentLabel</i> attribute), 196	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Items</i> attribute), 145
plugin_multi_attrib (<i>slixmpp.plugins.xep_0363.stanza.Header</i> attribute), 210	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Publish</i> attribute), 145
plugin_multi_attrib (<i>slixmpp.plugins.xep_0394.stanza.BlockCode</i> attribute), 216	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub</i> attribute), 146
plugin_multi_attrib (<i>slixmpp.plugins.xep_0394.stanza.BlockQuote</i> attribute), 216	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Retract</i> attribute), 146
plugin_multi_attrib (<i>slixmpp.plugins.xep_0394.stanza.Li</i> attribute), 217	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe</i> attribute), 147
plugin_multi_attrib (<i>slixmpp.plugins.xep_0394.stanza.List</i> attribute), 217	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe</i> attribute), 147
plugin_multi_attrib (<i>slixmpp.plugins.xep_0394.stanza.Span</i> attribute), 218	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.Event</i> attribute), 152
plugin_multi_attrib (<i>slixmpp.xmlstream.stanzabase.ElementBase</i> attribute), 85	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_event.Event</i> attribute), 152
plugin_overrides (<i>slixmpp.plugins.xep_0013.stanza.Offline</i>	at-	plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 149
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 149
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 149
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 150
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 150
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 151
		plugin_overrides (<i>slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner</i> attribute), 151

attribute), 151
plugin_overrides (slixmpp.plugins.xep_0065.stanza.Socks5in_tag_map (slixmpp.plugins.xep_0054.stanza.Logo attribute), 155
plugin_overrides (slixmpp.plugins.xep_0079.stanza.AMPin_tag_map (slixmpp.plugins.xep_0054.stanza.Photo attribute), 159
plugin_overrides (slixmpp.plugins.xep_0079.stanza.FailedRulestag_map (slixmpp.plugins.xep_0054.stanza.Sound attribute), 160
plugin_overrides (slixmpp.plugins.xep_0079.stanza.InvalidRulesTag_map (slixmpp.plugins.xep_0054.stanza.VCardTemp attribute), 160
plugin_overrides (slixmpp.plugins.xep_0079.stanza.UnsupportedActionsTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Affiliations attribute), 160
plugin_overrides (slixmpp.plugins.xep_0079.stanza.UnsupportedConfigurationsTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Configure attribute), 160
plugin_overrides (slixmpp.plugins.xep_0084.stanza.MetaDataTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Items attribute), 165
plugin_overrides (slixmpp.plugins.xep_0152.stanza.RetractabilityTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Publish attribute), 175
plugin_overrides (slixmpp.plugins.xep_0221.stanza.MediaTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Pubsub attribute), 188
plugin_overrides (slixmpp.plugins.xep_0257.stanza.GetTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Retract attribute), 194
plugin_overrides (slixmpp.plugins.xep_0258.stanza.SubscribeTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe attribute), 195
plugin_overrides (slixmpp.plugins.xep_0258.stanza.UnsubscribeTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Subscription attribute), 195
plugin_overrides (slixmpp.plugins.xep_0258.stanza.ExpiryLabelTag_map (slixmpp.plugins.xep_0060.stanza.pubsub.Subscription attribute), 196
plugin_overrides (slixmpp.plugins.xep_0258.stanza.LabelTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_errors.Publish attribute), 196
plugin_overrides (slixmpp.plugins.xep_0258.stanza.SpecifyLabelTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute), 196
plugin_overrides (slixmpp.plugins.xep_0394.stanza.PluginTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute), 217
plugin_overrides (slixmpp.plugins.xep_0394.stanza.MarkTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute), 217
plugin_overrides (slixmpp.plugins.xep_0394.stanza.SpanTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_event.Event attribute), 218
plugin_overrides (slixmpp.xmlstream.stanzabase.ElementBaseTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Default attribute), 85
plugin_tag_map (slixmpp.plugins.xep_0004.stanza.fieldFormFieldTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute), 101
plugin_tag_map (slixmpp.plugins.xep_0009.stanza.RPCMethodCallTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute), 104
plugin_tag_map (slixmpp.plugins.xep_0009.stanza.RPCMethodResponseTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute), 104
plugin_tag_map (slixmpp.plugins.xep_0009.stanza.RPCMethodQueryTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute), 105
plugin_tag_map (slixmpp.plugins.xep_0013.stanza.OfflinePluginTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Owner attribute), 106
plugin_tag_map (slixmpp.plugins.xep_0030.stanza.items.DisjointMetaTag_map (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Publish attribute), 117
plugin_tag_map (slixmpp.plugins.xep_0033.stanza.AddresseesTag_map (slixmpp.plugins.xep_0065.stanza.Socks5 attribute), 119
plugin_tag_map (slixmpp.plugins.xep_0054.stanza.AgentPluginTag_map (slixmpp.plugins.xep_0079.stanza.AMP attribute), 155

```

        attribute), 159
plugin_tag_map (slixmpp.plugins.xep_0079.stanza.FailedRule) (slixmpp.xmlstream.handler.Waiter method),
        attribute), 160                                         91
plugin_tag_map (slixmpp.plugins.xep_0079.stanza.InvalidRule (class in slixmpp.stanza), 229
        attribute), 160                                         Presence () (slixmpp.basexmpp.BaseXMPP method),
plugin_tag_map (slixmpp.plugins.xep_0079.stanza.UnsupportedActions
        attribute), 160                                         presence_available, 64
plugin_tag_map (slixmpp.plugins.xep_0079.stanza.UnsupportedCondition, 64
        attribute), 161                                         presence_form, 64
plugin_tag_map (slixmpp.plugins.xep_0084.stanza.MetaData_probe, 64
        attribute), 165                                         presence_subscribe, 64
plugin_tag_map (slixmpp.plugins.xep_0122.stanza.FormValidation_subscribed, 64
        attribute), 173                                         presence_unavailable, 64
plugin_tag_map (slixmpp.plugins.xep_0152.stanza.Reachability_unsubscribe, 65
        attribute), 175                                         presence_unsubscribed, 65
plugin_tag_map (slixmpp.plugins.xep_0221.stanza.Media_privateCarbon (class
        attribute), 188                                         slixmpp.plugins.xep_0280.stanza), 198
in
plugin_tag_map (slixmpp.plugins.xep_0257.stanza.Certificate_XML (class
        attribute), 194                                         slixmpp.plugins.xep_0049.stanza), 126
in
plugin_tag_map (slixmpp.plugins.xep_0258.stanza.Catalog_oceed (class in slixmpp.plugins.xep_0353.stanza),
        attribute), 195                                         209
plugin_tag_map (slixmpp.plugins.xep_0258.stanza.CatalogItems () (slixmpp.basexmpp.BaseXMPP method),
        attribute), 195                                         72
plugin_tag_map (slixmpp.plugins.xep_0258.stanza.EquivalentLabel (slixmpp.xmlstream.xmlstream.XMLStream
        attribute), 196                                         method), 97
plugin_tag_map (slixmpp.plugins.xep_0258.stanza.Label_prodID (class in slixmpp.plugins.xep_0054.stanza), 135
        attribute), 196                                         Propose (class in slixmpp.plugins.xep_0353.stanza),
plugin_tag_map (slixmpp.plugins.xep_0258.stanza.SecurityLabel209
        attribute), 196                                         proxy_config (slixmpp.xmlstream.xmlstream.XMLStream
plugin_tag_map (slixmpp.plugins.xep_0394.stanza.List attribute), 97
        attribute), 217                                         Publish (class in slixmpp.plugins.xep_0060.stanza.pubsub),
plugin_tag_map (slixmpp.plugins.xep_0394.stanza.Markup 145
        attribute), 217                                         publish () (slixmpp.plugins.xep_0060.XEP_0060
plugin_tag_map (slixmpp.plugins.xep_0394.stanza.Span method), 142
        attribute), 218                                         publish () (slixmpp.plugins.xep_0163.XEP_0163
plugin_tag_map (slixmpp.xmlstream.stanzabase.ElementBase method), 176
        attribute), 85                                         publish_activity ()
plugin_whitelist (slixmpp.basexmpp.BaseXMPP (slixmpp.plugins.xep_0108.XEP_0108
        attribute), 72                                         method), 170
plugins (slixmpp.xmlstream.stanzabase.ElementBase publish_gaming () (slixmpp.plugins.xep_0196.XEP_0196
        attribute), 85                                         method), 180
Pointer (class in slixmpp.plugins.xep_0084.stanza), 165                                         publish_location ()
pop () (slixmpp.xmlstream.stanzabase.ElementBase
        method), 86                                         (slixmpp.plugins.xep_0080.XEP_0080
method), 161                                         method), 161
Preferences (class in slixmpp.plugins.xep_0313.stanza), 201                                         publish_mood () (slixmpp.plugins.xep_0107.XEP_0107
method), 169                                         method), 169
prep_handlers () (slixmpp.plugins.xep_0050.XEP_0050
        method), 127                                         publish_nick () (slixmpp.plugins.xep_0172.XEP_0172
method), 177                                         method), 177
prerun () (slixmpp.xmlstream.handler.base.BaseHandler
        method), 89                                         publish_reachability ()
prerun () (slixmpp.xmlstream.handler.Callback publish_tune () (slixmpp.plugins.xep_0118.XEP_0118
        method), 89                                         method), 171
prerun () (slixmpp.xmlstream.handler CoroutineCallbackPublishOptions (class
        method), 89                                         in

```

```

    slixmpp.plugins.xep_0060.stanza.pubsub),           register_plugins() (in module
    145                                         slixmpp.plugins.xep_0403.stanza), 218
Pubsub (class in slixmpp.plugins.xep_0060.stanza.pubsub) register_plugins() (in module
    146                                         slixmpp.plugins.xep_0404.stanza), 219
pubsub_config, 65 register_plugins() (in module
pubsub_delete, 65                                         slixmpp.plugins.xep_0405.stanza), 220
pubsub_publish, 65 register_plugins() (in module
pubsub_purge, 65                                         slixmpp.plugins.xep_0422.stanza), 222
pubsub_retract, 65 register_plugins() (in module
pubsub_subscription, 65                                         slixmpp.plugins.xep_0424.stanza), 223
PubsubErrorCondition (class in slixmpp.plugins.xep_0060.stanza.pubsub_errors), register_plugins() (in module
    148                                         slixmpp.plugins.xep_0425.stanza), 224
PubsubOwner (class in slixmpp.plugins.xep_0060.stanza.pubsub_owner), register_plugins() (in module
    151                                         slixmpp.plugins.xep_0428.stanza), 224
purge() (slixmpp.plugins.xep_0060.XEP_0060 method), 142 register_plugins() (in module
    142                                         slixmpp.plugins.xep_0437.stanza), 225
Put (class in slixmpp.plugins.xep_0363.stanza), 210 register_plugins() (in module
    210                                         slixmpp.plugins.xep_0439.stanza), 226
R register_plugins() (in module
    72                                         (slixmpp.basexmpp.BaseXMPP method),
RAI (class in slixmpp.plugins.xep_0437.stanza), 225 register_stanza()
Reachability (class in slixmpp.plugins.xep_0152.stanza), 175 register_stanza() (in module
    175                                         (slixmpp.xmlstream.xmlstream.XMLStream
Reaction (class in slixmpp.plugins.xep_0444.stanza), method), 97
    227 register_stanza_plugin() (in module
reactions, 65                                         slixmpp.xmlstream.stanzabase), 77
Reactions (class in slixmpp.plugins.xep_0444.stanza), register_url_handler()
    227                                         (slixmpp.plugins.xep_0066.XEP_0066
receipt_received, 65 remove_handler() (slixmpp.xmlstream.xmlstream.XMLStream
Received (class in slixmpp.plugins.xep_0184.stanza), method), 97
    178 remove_interest()
Received (class in slixmpp.plugins.xep_0333.stanza), 206 remove_interest() (slixmpp.plugins.xep_0163.XEP_0163
ReceivedCarbon (class in slixmpp.plugins.xep_0280.stanza), 198 method), 176
remove_stanza()
reconnect () (slixmpp.xmlstream.xmlstream.XMLStream remove_stanza() (slixmpp.xmlstream.xmlstream.XMLStream
    method), 97 remove_stanza() (slixmpp.xmlstream.xmlstream.XMLStream
Register (class in slixmpp.plugins.xep_0077.stanza), method), 97
    158 Replace (class in slixmpp.plugins.xep_0308.stanza),
register_feature()                                         200
    67 reply()
        (slixmpp.clientxmpp.ClientXMPP method), 103
register_handler()                                         reply() (slixmpp.plugins.xep_0004.stanza.form.Form
    97 reply() (slixmpp.stanza.Iq method), 231
register_pep() (slixmpp.plugins.xep_0163.XEP_0163 reply() (slixmpp.stanza.Message method), 229
    method), 176 reply() (slixmpp.stanza.Presence method), 230
register_plugin() (slixmpp.basexmpp.BaseXMPP reply() (slixmpp.xmlstream.stanzabase.StanzaBase
    method), 72 method), 88
register_plugins() (in module Report (class in slixmpp.plugins.xep_0377.stanza), 215
    slixmpp.plugins.xep_0359.stanza), 210 Request (class in slixmpp.plugins.xep_0184.stanza),
register_plugins() (in module                                         178
    slixmpp.plugins.xep_0369.stanza), 214 Request (class in slixmpp.plugins.xep_0363.stanza),

```

```

210
request_ack () (slixmpp.plugins.xep_0198.XEP_0198
    method), 181
request_attention () (slixmpp.plugins.xep_0224.XEP_0224
    method), 190
RequestAck (class in slixmpp.plugins.xep_0198.stanza), 183
requested_jid (slixmpp.basexmpp.BaseXMPP attribute), 72
resource () (slixmpp.basexmpp.BaseXMPP property), 73
Response (class in slixmpp.plugins.xep_0439.stanza), 226
restore_defaults () (slixmpp.plugins.xep_0030.XEP_0030
    method), 112
Result (class in slixmpp.plugins.xep_0313.stanza), 202
Resume (class in slixmpp.plugins.xep_0198.stanza), 183
Resumed (class in slixmpp.plugins.xep_0198.stanza), 184
Retract (class in slixmpp.plugins.xep_0060.stanza.pubsub), 146
Retract (class in slixmpp.plugins.xep_0353.stanza), 209
Retract (class in slixmpp.plugins.xep_0424.stanza), 223
retract () (slixmpp.plugins.xep_0060.XEP_0060
    method), 142
Retracted (class in slixmpp.plugins.xep_0424.stanza), 223
retrieve () (slixmpp.plugins.xep_0222.XEP_0222
    method), 188
retrieve () (slixmpp.plugins.xep_0223.XEP_0223
    method), 189
retrieve () (slixmpp.plugins.xep_0313.XEP_0313
    method), 200
Rev (class in slixmpp.plugins.xep_0054.stanza), 136
RevokeCert (class in slixmpp.plugins.xep_0257.stanza), 194
Role (class in slixmpp.plugins.xep_0054.stanza), 136
room_activity, 65
room_activity_bare, 65
RootStanza (class in slixmpp.stanza.rootstanza), 227
roster (slixmpp.basexmpp.BaseXMPP attribute), 73
roster_update, 66
RPCQuery (class in slixmpp.plugins.xep_0009.stanza.RPC), 104
Rule (class in slixmpp.plugins.xep_0079.stanza), 160
run () (slixmpp.xmlstream.handler.base.BaseHandler
    method), 89
run () (slixmpp.xmlstream.handler.Callback method), 89
run () (slixmpp.xmlstream.handler.CoroutineCallback
    method), 90
run () (slixmpp.xmlstream.handler.Waiter method), 91
run_filters () (slixmpp.xmlstream.xmlstream.XMLStream
    method), 98
S
schedule () (slixmpp.xmlstream.xmlstream.XMLStream
    method), 98
SecurityLabel (class in slixmpp.plugins.xep_0258.stanza), 196
send () (slixmpp.stanza.Iq method), 231
send () (slixmpp.xmlstream.stanzabase.StanzaBase
    method), 88
send () (slixmpp.xmlstream.xmlstream.XMLStream
    method), 98
send_ack () (slixmpp.plugins.xep_0198.XEP_0198
    method), 181
send_active () (slixmpp.plugins.xep_0352.XEP_0352
    method), 207
send_affiliation_list () (slixmpp.plugins.xep_0045.XEP_0045
    method), 121
send_command () (slixmpp.plugins.xep_0050.XEP_0050
    method), 127
send_inactive () (slixmpp.plugins.xep_0352.XEP_0352
    method), 207
send_invitation () (slixmpp.plugins.xep_0249.XEP_0249
    method), 192
send_marker () (slixmpp.plugins.xep_0333.XEP_0333
    method), 205
send_message () (slixmpp.basexmpp.BaseXMPP
    method), 73
send_oob () (slixmpp.plugins.xep_0066.XEP_0066
    method), 156
send_ping () (slixmpp.plugins.xep_0199.XEP_0199
    method), 185
send_presence () (slixmpp.basexmpp.BaseXMPP
    method), 73
send_presence_subscription () (slixmpp.basexmpp.BaseXMPP
    method), 73
send_raw () (slixmpp.xmlstream.xmlstream.XMLStream
    method), 98
send_reactions () (slixmpp.plugins.xep_0444.XEP_0444
    method), 226
send_retraction () (slixmpp.plugins.xep_0424.XEP_0424
    method), 222
send_role_list () (slixmpp.plugins.xep_0045.XEP_0045
    method), 121
send_xml () (slixmpp.xmlstream.xmlstream.XMLStream
    method), 98
sent_presence, 66

```

```

SentCarbon          (class      in  set_body () (slixmpp.plugins.xep_0071.stanza.XHTML_IM
                     slixmpp.plugins.xep_0280.stanza), 198   method), 157
sentpresence        (slixmpp.basexmpp.BaseXMPP at-  set_carbon_received()
                     tribute), 73                                (slixmpp.plugins.xep_0280.stanza.ReceivedCarbon
                                                               method), 198
server()           (slixmpp.basexmpp.BaseXMPP property), 73
session_end, 66
session_end()       (slixmpp.plugins.xep_0198.XEP_0198
                     method), 181
session_start, 66
Set (class in slixmpp.plugins.xep_0059.stanza), 139
set_abuse()         (slixmpp.plugins.xep_0377.stanza.Report
                     method), 215
set_accuracy()      (slixmpp.plugins.xep_0080.stanza.Geoloc
                     method), 163
set_actions()       (slixmpp.plugins.xep_0050.stanza.Command
                     method), 129
set_activities()    (slixmpp.plugins.xep_0437.stanza.RAI
                     method), 225
set_addresses()     (slixmpp.plugins.xep_0033.stanza.Addresses
                     method), 119
set_affiliation()  (slixmpp.plugins.xep_0045.stanza.MUCBase
                     method), 123
set_affiliation()  (slixmpp.plugins.xep_0045.XEP_0045
                     method), 121
set_algo()          (slixmpp.plugins.xep_0300.stanza.Hash
                     method), 199
set_all()           (slixmpp.plugins.xep_0033.stanza.Addresses
                     method), 119
set_alt()           (slixmpp.plugins.xep_0080.stanza.Geoloc
                     method), 163
set_always()        (slixmpp.plugins.xep_0313.stanza.Preferences
                     method), 202
set_answer()         (slixmpp.plugins.xep_0004.stanza.field.FormField
                     method), 102
set_attention()     (slixmpp.plugins.xep_0224.stanza.Attention
                     method), 190
set_backend()        (slixmpp.plugins.xep_0050.XEP_0050
                     method), 128
set_basic()          (slixmpp.plugins.xep_0122.stanza.FormValidation
                     method), 173
set_bcc()            (slixmpp.plugins.xep_0033.stanza.Addresses
                     method), 119
set_bday()           (slixmpp.plugins.xep_0054.stanza.Birthday
                     method), 131
set_bearing()        (slixmpp.plugins.xep_0080.stanza.Geoloc
                     method), 163
set_before()          (slixmpp.plugins.xep_0059.stanza.Set
                     method), 140
set_binval()          (slixmpp.plugins.xep_0054.stanza.BinVal
                     method), 131
set_block_size()     (slixmpp.plugins.xep_0047.stanza.Open
                     method), 125
in    set_body () (slixmpp.plugins.xep_0071.stanza.XHTML_IM
                     method), 157
at-  set_carbon_received()
                     (slixmpp.plugins.xep_0280.stanza.ReceivedCarbon
                     method), 198
set_carbon_sent()
                     (slixmpp.plugins.xep_0280.stanza.SentCarbon
                     method), 198
set_categories()
                     (slixmpp.plugins.xep_0054.stanza.Categories
                     method), 131
set_cc()
                     (slixmpp.plugins.xep_0033.stanza.Addresses
                     method), 119
set_cert_management()
                     (slixmpp.plugins.xep_0257.stanza.AppendCert
                     method), 193
set_chat_state()
                     (slixmpp.plugins.xep_0085.stanza.ChatState
                     method), 166
set_code()
                     (slixmpp.plugins.xep_0045.stanza.MUCStatus
                     method), 124
set_code()
                     (slixmpp.plugins.xep_0332.stanza.response.HTTPResponse
                     method), 205
set_condition()
                     (slixmpp.plugins.xep_0060.stanza.pubsub_errors.Pub
                     method), 148
set_condition()
                     (slixmpp.plugins.xep_0086.stanza.LegacyError
                     method), 168
set_config()
                     (slixmpp.plugins.xep_0060.stanza.pubsub_owner.Default
                     method), 149
set_data()
                     (slixmpp.plugins.xep_0047.stanza.Data
                     method), 125
set_data()
                     (slixmpp.plugins.xep_0231.stanza.BitsOfBinary
                     method), 191
set_data()
                     (slixmpp.plugins.xep_0332.stanza.data.HTTPData
                     method), 203
set_default()
                     (slixmpp.plugins.xep_0258.stanza.CatalogItem
                     method), 195
set_delivered()
                     (slixmpp.plugins.xep_0033.stanza.Address
                     method), 118
set_desc()
                     (slixmpp.plugins.xep_0054.stanza.Desc
                     method), 132
set_descriptions()
                     (slixmpp.plugins.xep_0353.stanza.Propose
                     method), 209
set_encrypted()
                     (slixmpp.plugins.xep_0027.stanza.Encrypted
                     method), 107
set_end()
                     (slixmpp.plugins.xep_0313.stanza.MAM
                     method), 201
set_error()
                     (slixmpp.plugins.xep_0080.stanza.Geoloc
                     method), 163
set_expiry()
                     (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventSu
                     method), 154
set_extended_info()
                     (slixmpp.plugins.xep_0128.XEP_0128
                     method), 173
set_false()
                     (slixmpp.plugins.xep_0004.stanza.field.FormField
                     method), 173

```

```

        method), 102
set_family () (slixmpp.plugins.xep_0054.stanza.Name set_jabberid () (slixmpp.plugins.xep_0054.stanza.JabberID
        method), 134
set_fault () (slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse (slixmpp.basexmpp.BaseXMPP method),
        method), 104
set_features () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfo () (slixmpp.plugins.xep_0013.stanza.Item
        method), 116
set_features () (slixmpp.plugins.xep_0030.XEP_0030.set_jid () (slixmpp.plugins.xep_0033.stanza.Address
        method), 112
set_fields () (slixmpp.plugins.xep_0004.stanza.form.Form.set_jid () (slixmpp.plugins.xep_0045.stanza.MUCBase
        method), 103
set_fields () (slixmpp.plugins.xep_0077.stanza.Register.set_jid () (slixmpp.plugins.xep_0060.stanza.pubsub.Affiliation
        method), 158
set_first_index () (slixmpp.plugins.xep_0059.stanza.Set method),
        140
set_from () (slixmpp.plugins.xep_0079.stanza.AMP
        method), 159
set_from () (slixmpp.plugins.xep_0203.stanza.Delay
        method), 187
set_from () (slixmpp.plugins.xep_0258.stanza.Catalog
        method), 195
set_from () (slixmpp.xmlstream.stanzabase.StanzaBase
        method), 88
set_given () (slixmpp.plugins.xep_0054.stanza.Name
        method), 134
set_h () (slixmpp.plugins.xep_0198.stanza.Ack
        method), 182
set_h () (slixmpp.plugins.xep_0198.stanza.Resume
        method), 183
set_h () (slixmpp.plugins.xep_0198.stanza.Resumed
        method), 184
set_headers () (slixmpp.plugins.xep_0131.stanza.Headers
        method), 174
set_identities () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfo.setmethod), 172
        method), 116
set_identities () (slixmpp.plugins.xep_0030.XEP_0030
        method), 112
set_info () (slixmpp.plugins.xep_0030.XEP_0030
        method), 112
set_instructions () (slixmpp.plugins.xep_0004.stanza.form.Form
        method), 103
set_ip_check () (slixmpp.plugins.xep_0279.stanza.IPCheck.setmax_items (), (slixmpp.plugins.xep_0060.stanza.pubsub.Items
        method), 197
set_item_attr () (slixmpp.plugins.xep_0045.stanza.MUCBase.setmessage (), (slixmpp.plugins.xep_0332.stanza.response.HTTPRespon
        method), 123
set_items () (slixmpp.plugins.xep_0004.stanza.form.Form.set_method (), (slixmpp.plugins.xep_0332.stanza.request.HTTPRequest
        method), 103
set_items () (slixmpp.plugins.xep_0030.stanza.items.DiscoItems.setmethod_name ())
        method), 117
set_items () (slixmpp.plugins.xep_0030.XEP_0030
        method), 112
set_items () (slixmpp.plugins.xep_0191.stanza.BlockList
        method), 134
        method), 180
set_jabberid () (slixmpp.plugins.xep_0054.stanza.JabberID
        method), 133
set_method_response () (slixmpp.basexmpp.BaseXMPP
        method), 73
set_options () (slixmpp.plugins.xep_0060.stanza.pubsub.Options
        method), 145
set_subscribe () (slixmpp.plugins.xep_0060.stanza.pubsub.Subscribe
        method), 147
set_subscription () (slixmpp.plugins.xep_0060.stanza.pubsub.Subscription
        method), 147
set_unsubscribe () (slixmpp.plugins.xep_0060.stanza.pubsub.Unsubscribe
        method), 148
set_event_subscr () (slixmpp.plugins.xep_0060.stanza.pubsub_event.EventSubscr
        method), 154
set_owner_redir () (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerRedir
        method), 150
set_owner_subscr () (slixmpp.plugins.xep_0060.stanza.pubsub_owner.OwnerSubscr
        method), 151
set_stream_host () (slixmpp.plugins.xep_0065.stanza.StreamHost
        method), 155
set_stream_host_used () (slixmpp.plugins.xep_0065.stanza.StreamHostUsed
        method), 155
set_geoloc () (slixmpp.plugins.xep_0080.stanza.Geoloc
        method), 163
set_user_tune () (slixmpp.plugins.xep_0118.stanza.UserTune
        method), 172
set_label () (slixmpp.plugins.xep_0054.stanza.Label
        method), 133
set_geoloc () (slixmpp.plugins.xep_0080.stanza.Geoloc
        method), 163
set_mailer () (slixmpp.plugins.xep_0054.stanza.Mailer
        method), 134
set_bits_of_binary () (slixmpp.plugins.xep_0231.stanza.BitsOfBinary
        method), 191
set_items () (slixmpp.plugins.xep_0060.stanza.pubsub.Items
        method), 145
set_message () (slixmpp.plugins.xep_0332.stanza.response.HTTPRespon
        method), 205
set_request () (slixmpp.plugins.xep_0332.stanza.request.HTTPRequest
        method), 204
set_method_call () (slixmpp.plugins.xep_0009.stanza.RPC.MethodCall
        method), 104
set_name () (slixmpp.plugins.xep_0054.stanza.Name
        method), 134

```

set_mucnick() (*slixmpp.stanza.Message method*), set_photo() (*slixmpp.plugins.xep_0153.stanza.VCardTempUpdate method*), 176
 229
 set_mucroom() (*slixmpp.stanza.Message method*), set_preferences() (*slixmpp.plugins.xep_0404.XEP_0404 method*), 219
 229
 set_multi() (*in module slixmpp.plugins.xep_0033.stanza*), 119 set_prefix() (*slixmpp.plugins.xep_0054.stanza.Name method*), 134
 set_never() (*slixmpp.plugins.xep_0313.stanza.Preferences method*), 134
 202 set_priority() (*slixmpp.stanza.Presence method*), 230
 set_nick() (*slixmpp.plugins.xep_0045.stanza.MUCBase method*), 123 set_prodid() (*slixmpp.plugins.xep_0054.stanza.ProdID method*), 135
 set_nick() (*slixmpp.plugins.xep_0172.stanza.UserNick method*), 178 set_publish_options() (*slixmpp.plugins.xep_0060.stanza.pubsub.PublishOptions method*), 146
 set_nick() (*slixmpp.plugins.xep_0369.XEP_0369 method*), 212 set_nickname() (*slixmpp.plugins.xep_0054.stanza.NickNameQuery method*), 232
 134 set_range() (*slixmpp.plugins.xep_0122.stanza.FormValidation method*), 173
 set_node_handler() (*slixmpp.plugins.xep_0030.XEP_0030 method*), 113 set_rating() (*slixmpp.plugins.xep_0118.stanza.UserTune method*), 172
 set_noreply() (*slixmpp.plugins.xep_0033.stanza.AddressesReactions method*), 226
 119 set_receipt() (*slixmpp.plugins.xep_0184.stanza.Received method*), 178
 set_notes() (*slixmpp.plugins.xep_0050.stanza.CommandSet method*), 130 set_redirect() (*slixmpp.plugins.xep_0060.stanza.pubsub_event.Event method*), 153
 set_notify() (*slixmpp.plugins.xep_0060.stanza.pubsubRetract method*), 146 set_registered() (*slixmpp.plugins.xep_0077.stanza.Register method*), 158
 137 set_remove() (*slixmpp.plugins.xep_0122.stanza.FormValidation method*), 158
 set_optional() (*slixmpp.plugins.xep_0198.stanza.StreamManagement method*), 119
 184 set_remove() (*slixmpp.plugins.xep_0033.stanza.Addresses method*), 119
 set_options() (*slixmpp.plugins.xep_0004.stanza.field.FormField method*), 102 set_reported() (*slixmpp.plugins.xep_0004.stanza.form.Form method*), 103
 set_options() (*slixmpp.plugins.xep_0060.stanza.pubsubOptions method*), 145 set_request() (*slixmpp.plugins.xep_0184.stanza.Request method*), 179
 set_orgunits() (*slixmpp.plugins.xep_0054.stanza.Orgset method*), 135 set_receipt() (*slixmpp.plugins.xep_0184.stanza.Request method*), 179
 set_params() (*slixmpp.plugins.xep_0009.stanza.RPC.MethodCall method*), 179 set_required() (*slixmpp.plugins.xep_0004.stanza.field.FormField method*), 104
 104 set_required() (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse method*), 102 set_required() (*slixmpp.plugins.xep_0060.stanza.base.OptionalSetting method*), 104
 set_parent_thread() (*slixmpp.stanza.Message method*), 229 set_required() (*slixmpp.plugins.xep_0198.stanza.StreamManagement method*), 143
 set_payload() (*slixmpp.plugins.xep_0060.stanza.pubsub.Item method*), 184 set_resource() (*slixmpp.plugins.xep_0332.stanza.request.HTTPRequest method*), 145
 set_payload() (*slixmpp.plugins.xep_0060.stanza.pubsub_event.ErrorEvent method*), 204 set_restrict() (*slixmpp.plugins.xep_0258.stanza.Catalog method*), 195
 set_payload() (*slixmpp.stanza.Iq method*), 232 set_results() (*slixmpp.plugins.xep_0013.stanza.Offline method*), 106
 88 set_per_hop() (*slixmpp.plugins.xep_0079.stanza.AMP method*), 159 set_results() (*slixmpp.plugins.xep_0313.stanza.MAM method*), 201

```

set_resume() (slixmpp.plugins.xep_0198.stanza.Enable           method), 134
              method), 182
set_resume() (slixmpp.plugins.xep_0198.stanza.Enabled         method), 187
              method), 182
set_rev()    (slixmpp.plugins.xep_0054.stanza.Rev            method), 186
              method), 136
set_role()   (slixmpp.plugins.xep_0045.stanza.MUCBase        method), 164
              method), 123
set_role()   (slixmpp.plugins.xep_0045.XEP_0045           method), 138
              method), 121
set_role()   (slixmpp.plugins.xep_0054.stanza.Role          method), 119
              method), 136
set_room()   (slixmpp.plugins.xep_0045.stanza.MUCBase        method), 195
              method), 123
set_room_config() (slixmpp.plugins.xep_0045.XEP_0045       method), 88
                  method), 121
set_seconds() (slixmpp.plugins.xep_0012.stanza.LastActivity true () (slixmpp.plugins.xep_0004.stanza.field.FormField
                  method), 105                                     method), 102
set_seq()    (slixmpp.plugins.xep_0047.stanza.Data          method), 230
              method), 125
set_shell()  (slixmpp.plugins.xep_0422.stanza.ApplyTo        set_type () (slixmpp.plugins.xep_0004.stanza.form.Form
              method), 222                                     method), 103
set_show()   (slixmpp.stanza.Presence method), 230
set_signed() (slixmpp.plugins.xep_0027.stanza.Signed        set_type () (slixmpp.stanza.Presence method), 230
              method), 88
set_since()  (slixmpp.plugins.xep_0319.stanza.Idle         set_type () (slixmpp.xmlstream.stanzabase.StanzaBase
              method), 202                                     method), 218
set_sort_string() (slixmpp.plugins.xep_0054.stanza.SortString set_type () (slixmpp.plugins.xep_0394.stanza.Span
                  method), 136                                     method), 137
set_spam()   (slixmpp.plugins.xep_0377.stanza.Report       set_tz () (slixmpp.plugins.xep_0054.stanza.TimeZone
              method), 215                                     method), 186
set_speed()  (slixmpp.plugins.xep_0080.stanza.Geoloc        set_tzo () (slixmpp.plugins.xep_0202.stanza.EntityTime
              method), 163                                     method), 199
set_stamp()  (slixmpp.plugins.xep_0203.stanza.Delay        set_uid () (slixmpp.plugins.xep_0054.stanza.UID
              method), 187                                     method), 138
set_stanza() (slixmpp.plugins.xep_0297.stanza.Forwarded     set_unsupported()
              method), 199                                     (slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorCo
set_stanza_values() (slixmpp.xmlstream.stanzabase.ElementBase set_url () (slixmpp.plugins.xep_0054.stanza.URL
                  method), 86                                     method), 138
set_start()  (slixmpp.plugins.xep_0313.stanza.MAM          set_users () (slixmpp.plugins.xep_0257.stanza.CertItem
              method), 201                                     method), 193
set_start()  (slixmpp.plugins.xep_0394.stanza.Li           set_utc () (slixmpp.plugins.xep_0202.stanza.EntityTime
              method), 217                                     method), 186
set_status() (slixmpp.plugins.xep_0012.stanza.LastActivity set_value () (slixmpp.plugins.xep_0004.stanza.field.FormField
              method), 105                                     method), 102
set_status_codes() (slixmpp.plugins.xep_0045.stanza.MUCBase set_value () (slixmpp.plugins.xep_0084.stanza.Data
                  method), 123                                     method), 164
set_subject() (slixmpp.plugins.xep_0045.XEP_0045           set_value () (slixmpp.plugins.xep_0107.stanza.UserMood
              method), 121                                     method), 169
set_suffix() (slixmpp.plugins.xep_0054.stanza.Name         set_value () (slixmpp.plugins.xep_0108.stanza.UserActivity
              method), 170                                     method), 170
set_value()  (slixmpp.plugins.xep_0221.stanza.URI          set_value () (slixmpp.plugins.xep_0221.stanza.URI
              method), 170

```

```

        method), 188
set_value () (slixmpp.plugins.xep_0258.stanza.DisplayMarking method), 131
        setup () (slixmpp.plugins.xep_0054.stanza.BinVal
method), 195
        setup () (slixmpp.plugins.xep_0054.stanza.Telephone
set_value () (slixmpp.plugins.xep_0258.stanza.ESSLabel method), 137
        setup () (slixmpp.plugins.xep_0060.stanza.pubsub_errors.PubsubErrorC
method), 196
        setup () (slixmpp.plugins.xep_0300.stanza.Hash
method), 148
        setup () (slixmpp.plugins.xep_0085.stanza.ChatState
set_value () (slixmpp.plugins.xep_0335.stanza.JSON_Container method), 166
        setup () (slixmpp.plugins.xep_0086.stanza.LegacyError
method), 207
        setup () (slixmpp.plugins.xep_0184.stanza.Received
set_value () (slixmpp.plugins.xep_0363.stanza.Header
method), 168
        setup () (slixmpp.plugins.xep_0184.stanza.Request
method), 210
        setup () (slixmpp.plugins.xep_0184.stanza.Received
set_value () (slixmpp.plugins.xep_0444.stanza.Reaction
method), 178
        setup () (slixmpp.plugins.xep_0184.stanza.Request
method), 227
        setup () (slixmpp.plugins.xep_0198.stanza.Ack
set_values () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 179
        setup () (slixmpp.plugins.xep_0198.stanza.Ack
method), 103
        setup () (slixmpp.plugins.xep_0198.stanza.Enable
set_values () (slixmpp.plugins.xep_0444.stanza.Reactions
method), 182
        setup () (slixmpp.plugins.xep_0198.stanza.Enable
method), 227
        setup () (slixmpp.plugins.xep_0332.stanza.request.HTTPRequest
method), 182
        setup () (slixmpp.plugins.xep_0198.stanza.Enabled
set_version () (slixmpp.plugins.xep_0332.stanza.response.HTTPResponse
method), 182
        setup () (slixmpp.plugins.xep_0198.stanza.Failed
method), 205
        setup () (slixmpp.plugins.xep_0198.stanza.RequestAck
set_with () (slixmpp.plugins.xep_0313.stanza.MAM
method), 183
        setup () (slixmpp.plugins.xep_0198.stanza.RequestAck
method), 201
        setup () (slixmpp.plugins.xep_0198.stanza.Resume
setAnswer () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 183
        setup () (slixmpp.plugins.xep_0198.stanza.Resume
method), 101
        setup () (slixmpp.plugins.xep_0198.stanza.Resumed
method), 93
        setup () (slixmpp.plugins.xep_0198.stanza.Resumed
setFalse () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 184
        setup () (slixmpp.plugins.xep_0224.stanza.Attention
method), 101
        setup () (slixmpp.plugins.xep_0224.stanza.Attention
setFields () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 190
        setup () (slixmpp.plugins.xep_0313.stanza.MAM
method), 103
        setup () (slixmpp.plugins.xep_0313.stanza.MAM
setInstructions ()
(slixmpp.plugins.xep_0004.stanza.form.Form
method), 201
        setup () (slixmpp.plugins.xep_0352.stanza.Active
method), 103
        setup () (slixmpp.plugins.xep_0352.stanza.Inactive
method), 208
Setnick (class in slixmpp.plugins.xep_0369.stanza), 214
        setup () (slixmpp.plugins.xep_0352.stanza.Inactive
method), 208
setOptions () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
        setup () (slixmpp.xmlstream.stanzabase.ElementBase
method), 86
setReported () (slixmpp.plugins.xep_0004.stanza.form.FormValue
method), 103
        setup () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 103
setRequired () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
        setup () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 103
setTrue () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 101
        setup () (slixmpp.plugins.xep_0027.stanza), 107
        module, 69
setup () (slixmpp.plugins.xep_0004.stanza.field.FormField
method), 102
        module, 69
        slixmpp.clientxmpp
setup () (slixmpp.plugins.xep_0004.stanza.form.Form
method), 103
        module, 67
        slixmpp.componentxmpp
setup () (slixmpp.plugins.xep_0013.stanza.Offline
method), 106
        module, 68
        slixmpp.exceptions
setup () (slixmpp.plugins.xep_0030.stanza.info.DiscoInfo
method), 116
        module, 74
        slixmpp.jid
setup () (slixmpp.plugins.xep_0030.stanza.items.DiscoItems
method), 118
        module, 75
        slixmpp.plugins.xep_0004

```

```
    module, 100
slixmpp.plugins.xep_0004.stanza.field
    module, 100
slixmpp.plugins.xep_0004.stanza.form
    module, 102
slixmpp.plugins.xep_0009
    module, 104
slixmpp.plugins.xep_0009.stanza.RPC
    module, 104
slixmpp.plugins.xep_0012
    module, 105
slixmpp.plugins.xep_0012.stanza
    module, 105
slixmpp.plugins.xep_0013
    module, 105
slixmpp.plugins.xep_0013.stanza
    module, 106
slixmpp.plugins.xep_0020
    module, 107
slixmpp.plugins.xep_0020.stanza
    module, 107
slixmpp.plugins.xep_0027
    module, 107
slixmpp.plugins.xep_0027.stanza
    module, 107
slixmpp.plugins.xep_0030
    module, 108
slixmpp.plugins.xep_0030.stanza.info
    module, 114
slixmpp.plugins.xep_0030.stanza.items
    module, 116
slixmpp.plugins.xep_0033
    module, 118
slixmpp.plugins.xep_0033.stanza
    module, 118
slixmpp.plugins.xep_0045
    module, 120
slixmpp.plugins.xep_0045.stanza
    module, 121
slixmpp.plugins.xep_0047
    module, 125
slixmpp.plugins.xep_0047.stanza
    module, 125
slixmpp.plugins.xep_0049
    module, 126
slixmpp.plugins.xep_0049.stanza
    module, 126
slixmpp.plugins.xep_0050
    module, 126
slixmpp.plugins.xep_0050.stanza
    module, 128
slixmpp.plugins.xep_0054
    module, 130
slixmpp.plugins.xep_0054.stanza
    module, 130
slixmpp.plugins.xep_0059
    module, 139
slixmpp.plugins.xep_0059.stanza
    module, 139
slixmpp.plugins.xep_0060
    module, 140
slixmpp.plugins.xep_0060.stanza.base
    module, 143
slixmpp.plugins.xep_0060.stanza.pubsub
    module, 143
slixmpp.plugins.xep_0060.stanza.pubsub_errors
    module, 148
slixmpp.plugins.xep_0060.stanza.pubsub_event
    module, 151
slixmpp.plugins.xep_0060.stanza.pubsub_owner
    module, 148
slixmpp.plugins.xep_0065
    module, 154
slixmpp.plugins.xep_0065.stanza
    module, 155
slixmpp.plugins.xep_0066
    module, 156
slixmpp.plugins.xep_0066.stanza
    module, 156
slixmpp.plugins.xep_0070
    module, 157
slixmpp.plugins.xep_0070.stanza
    module, 157
slixmpp.plugins.xep_0071
    module, 157
slixmpp.plugins.xep_0071.stanza
    module, 157
slixmpp.plugins.xep_0077
    module, 158
slixmpp.plugins.xep_0077.stanza
    module, 158
slixmpp.plugins.xep_0079
    module, 159
slixmpp.plugins.xep_0079.stanza
    module, 159
slixmpp.plugins.xep_0080
    module, 161
slixmpp.plugins.xep_0080.stanza
    module, 162
slixmpp.plugins.xep_0082
    module, 164
slixmpp.plugins.xep_0084
    module, 164
slixmpp.plugins.xep_0084.stanza
    module, 164
slixmpp.plugins.xep_0085
    module, 165
slixmpp.plugins.xep_0085.stanza
    module, 165
```

```

    module, 166
slixmpp.plugins.xep_0086
    module, 167
slixmpp.plugins.xep_0086.stanza
    module, 167
slixmpp.plugins.xep_0092
    module, 168
slixmpp.plugins.xep_0092.stanza
    module, 168
slixmpp.plugins.xep_0106
    module, 169
slixmpp.plugins.xep_0107
    module, 169
slixmpp.plugins.xep_0107.stanza
    module, 169
slixmpp.plugins.xep_0108
    module, 170
slixmpp.plugins.xep_0108.stanza
    module, 170
slixmpp.plugins.xep_0115
    module, 171
slixmpp.plugins.xep_0115.stanza
    module, 171
slixmpp.plugins.xep_0118
    module, 171
slixmpp.plugins.xep_0118.stanza
    module, 172
slixmpp.plugins.xep_0122
    module, 172
slixmpp.plugins.xep_0122.stanza
    module, 172
slixmpp.plugins.xep_0128
    module, 173
slixmpp.plugins.xep_0131
    module, 174
slixmpp.plugins.xep_0131.stanza
    module, 174
slixmpp.plugins.xep_0133
    module, 174
slixmpp.plugins.xep_0152
    module, 174
slixmpp.plugins.xep_0152.stanza
    module, 175
slixmpp.plugins.xep_0153
    module, 175
slixmpp.plugins.xep_0153.stanza
    module, 175
slixmpp.plugins.xep_0163
    module, 176
slixmpp.plugins.xep_0172
    module, 177
slixmpp.plugins.xep_0172.stanza
    module, 177
slixmpp.plugins.xep_0184
    module, 178
slixmpp.plugins.xep_0184.stanza
    module, 178
slixmpp.plugins.xep_0186
    module, 179
slixmpp.plugins.xep_0186.stanza
    module, 179
slixmpp.plugins.xep_0191
    module, 180
slixmpp.plugins.xep_0191.stanza
    module, 180
slixmpp.plugins.xep_0196
    module, 180
slixmpp.plugins.xep_0196.stanza
    module, 181
slixmpp.plugins.xep_0198
    module, 181
slixmpp.plugins.xep_0198.stanza
    module, 182
slixmpp.plugins.xep_0199
    module, 184
slixmpp.plugins.xep_0199.stanza
    module, 185
slixmpp.plugins.xep_0202
    module, 185
slixmpp.plugins.xep_0202.stanza
    module, 186
slixmpp.plugins.xep_0203
    module, 187
slixmpp.plugins.xep_0203.stanza
    module, 187
slixmpp.plugins.xep_0221
    module, 187
slixmpp.plugins.xep_0221.stanza
    module, 187
slixmpp.plugins.xep_0222
    module, 188
slixmpp.plugins.xep_0223
    module, 189
slixmpp.plugins.xep_0224
    module, 190
slixmpp.plugins.xep_0224.stanza
    module, 190
slixmpp.plugins.xep_0231
    module, 191
slixmpp.plugins.xep_0231.stanza
    module, 191
slixmpp.plugins.xep_0235
    module, 191
slixmpp.plugins.xep_0235.stanza
    module, 191
slixmpp.plugins.xep_0249
    module, 192
slixmpp.plugins.xep_0249.stanza
    module, 192

```

```
    module, 192
slixmpp.plugins.xep_0256
    module, 193
slixmpp.plugins.xep_0257
    module, 193
slixmpp.plugins.xep_0257.stanza
    module, 193
slixmpp.plugins.xep_0258
    module, 194
slixmpp.plugins.xep_0258.stanza
    module, 194
slixmpp.plugins.xep_0279
    module, 197
slixmpp.plugins.xep_0279.stanza
    module, 197
slixmpp.plugins.xep_0280
    module, 197
slixmpp.plugins.xep_0280.stanza
    module, 197
slixmpp.plugins.xep_0297
    module, 198
slixmpp.plugins.xep_0297.stanza
    module, 199
slixmpp.plugins.xep_0300
    module, 199
slixmpp.plugins.xep_0300.stanza
    module, 199
slixmpp.plugins.xep_0308
    module, 200
slixmpp.plugins.xep_0308.stanza
    module, 200
slixmpp.plugins.xep_0313
    module, 200
slixmpp.plugins.xep_0313.stanza
    module, 201
slixmpp.plugins.xep_0319
    module, 202
slixmpp.plugins.xep_0319.stanza
    module, 202
slixmpp.plugins.xep_0332
    module, 202
slixmpp.plugins.xep_0332.stanza.data
    module, 203
slixmpp.plugins.xep_0332.stanza.request
    module, 203
slixmpp.plugins.xep_0332.stanza.responses
    module, 204
slixmpp.plugins.xep_0333
    module, 205
slixmpp.plugins.xep_0333.stanza
    module, 205
slixmpp.plugins.xep_0334
    module, 206
slixmpp.plugins.xep_0334.stanza
    module, 206
slixmpp.plugins.xep_0335
    module, 207
slixmpp.plugins.xep_0335.stanza
    module, 207
slixmpp.plugins.xep_0352
    module, 207
slixmpp.plugins.xep_0352.stanza
    module, 207
slixmpp.plugins.xep_0353
    module, 208
slixmpp.plugins.xep_0353.stanza
    module, 208
slixmpp.plugins.xep_0359
    module, 209
slixmpp.plugins.xep_0359.stanza
    module, 209
slixmpp.plugins.xep_0363
    module, 210
slixmpp.plugins.xep_0363.stanza
    module, 210
slixmpp.plugins.xep_0369
    module, 211
slixmpp.plugins.xep_0369.stanza
    module, 213
slixmpp.plugins.xep_0377
    module, 215
slixmpp.plugins.xep_0377.stanza
    module, 215
slixmpp.plugins.xep_0380
    module, 216
slixmpp.plugins.xep_0380.stanza
    module, 216
slixmpp.plugins.xep_0394
    module, 216
slixmpp.plugins.xep_0394.stanza
    module, 216
slixmpp.plugins.xep_0403
    module, 218
slixmpp.plugins.xep_0403.stanza
    module, 218
slixmpp.plugins.xep_0404
    module, 218
slixmpp.plugins.xep_0404.stanza
    module, 219
slixmpp.plugins.xep_0405
    module, 220
slixmpp.plugins.xep_0405.stanza
    module, 220
slixmpp.plugins.xep_0421
    module, 221
slixmpp.plugins.xep_0421.stanza
    module, 221
slixmpp.plugins.xep_0422
```

<pre> module, 221 slixmpp.plugins.xep_0422.stanza module, 222 slixmpp.plugins.xep_0424 module, 222 slixmpp.plugins.xep_0424.stanza module, 223 slixmpp.plugins.xep_0425 module, 223 slixmpp.plugins.xep_0425.stanza module, 223 slixmpp.plugins.xep_0428 module, 224 slixmpp.plugins.xep_0428.stanza module, 224 slixmpp.plugins.xep_0437 module, 224 slixmpp.plugins.xep_0437.stanza module, 224 slixmpp.plugins.xep_0439 module, 225 slixmpp.plugins.xep_0439.stanza module, 226 slixmpp.plugins.xep_0444 module, 226 slixmpp.plugins.xep_0444.stanza module, 227 slixmpp.stanza module, 229 slixmpp.stanza.rootstanza module, 227 slixmpp.xmlstream.handler module, 89 slixmpp.xmlstream.handler.base module, 88 slixmpp.xmlstream.matcher.base module, 91 slixmpp.xmlstream.matcher.id module, 92 slixmpp.xmlstream.matcher.stanzapath module, 92 slixmpp.xmlstream.matcher.xmlmask module, 93 slixmpp.xmlstream.matcher.xpath module, 92 slixmpp.xmlstream.stanzabase module, 76 slixmpp.xmlstream.xmlstream module, 93 Slot (<i>class in slixmpp.plugins.xep_0363.stanza</i>), 211 sm_disabled, 66 sm_enabled, 66 socket_error, 66 Socks5 (<i>class in slixmpp.plugins.xep_0065.stanza</i>), 155 </pre>	SortString <i>(class in slixmpp.plugins.xep_0054.stanza)</i> , 136 Sound (<i>class in slixmpp.plugins.xep_0054.stanza</i>), 136 Span (<i>class in slixmpp.plugins.xep_0394.stanza</i>), 217 specific (<i>slixmpp.plugins.xep_0108.stanza.UserActivity attribute</i>), 170 stanza, 233 stanza (<i>slixmpp.basexmpp.BaseXMPP attribute</i>), 74 stanza (<i>slixmpp.plugins.xep_0004.XEP_0004 attribute</i>), 100 stanza (<i>slixmpp.plugins.xep_0009.XEP_0009 attribute</i>), 104 stanza (<i>slixmpp.plugins.xep_0012.XEP_0012 attribute</i>), 105 stanza (<i>slixmpp.plugins.xep_0013.XEP_0013 attribute</i>), 105 stanza (<i>slixmpp.plugins.xep_0020.XEP_0020 attribute</i>), 107 stanza (<i>slixmpp.plugins.xep_0027.XEP_0027 attribute</i>), 107 stanza (<i>slixmpp.plugins.xep_0030.XEP_0030 attribute</i>), 113 stanza (<i>slixmpp.plugins.xep_0033.XEP_0033 attribute</i>), 118 stanza (<i>slixmpp.plugins.xep_0045.XEP_0045 attribute</i>), 121 stanza (<i>slixmpp.plugins.xep_0049.XEP_0049 attribute</i>), 126 stanza (<i>slixmpp.plugins.xep_0050.XEP_0050 attribute</i>), 128 stanza (<i>slixmpp.plugins.xep_0059.XEP_0059 attribute</i>), 139 stanza (<i>slixmpp.plugins.xep_0060.XEP_0060 attribute</i>), 142 stanza (<i>slixmpp.plugins.xep_0066.XEP_0066 attribute</i>), 156 stanza (<i>slixmpp.plugins.xep_0070.XEP_0070 attribute</i>), 157 stanza (<i>slixmpp.plugins.xep_0071.XEP_0071 attribute</i>), 157 stanza (<i>slixmpp.plugins.xep_0077.XEP_0077 attribute</i>), 158 stanza (<i>slixmpp.plugins.xep_0079.XEP_0079 attribute</i>), 159 stanza (<i>slixmpp.plugins.xep_0080.XEP_0080 attribute</i>), 161 stanza (<i>slixmpp.plugins.xep_0084.XEP_0084 attribute</i>), 164 stanza (<i>slixmpp.plugins.xep_0085.XEP_0085 attribute</i>), 165 stanza (<i>slixmpp.plugins.xep_0086.XEP_0086 attribute</i>), 167 stanza (<i>slixmpp.plugins.xep_0092.XEP_0092 attribute</i>), 168
---	--

stanza (<i>slixmpp.plugins.xep_0107.XEP_0107 tribute</i>), 169	at-	stanza (<i>slixmpp.plugins.xep_0313.XEP_0313 tribute</i>), 200	at-
stanza (<i>slixmpp.plugins.xep_0108.XEP_0108 tribute</i>), 170	at-	stanza (<i>slixmpp.plugins.xep_0319.XEP_0319 tribute</i>), 202	at-
stanza (<i>slixmpp.plugins.xep_0115.XEP_0115 tribute</i>), 171	at-	stanza (<i>slixmpp.plugins.xep_0333.XEP_0333 tribute</i>), 205	at-
stanza (<i>slixmpp.plugins.xep_0118.XEP_0118 tribute</i>), 171	at-	stanza (<i>slixmpp.plugins.xep_0334.XEP_0334 tribute</i>), 206	at-
stanza (<i>slixmpp.plugins.xep_0122.XEP_0122 tribute</i>), 172	at-	stanza (<i>slixmpp.plugins.xep_0335.XEP_0335 tribute</i>), 207	at-
stanza (<i>slixmpp.plugins.xep_0131.XEP_0131 tribute</i>), 174	at-	stanza (<i>slixmpp.plugins.xep_0352.XEP_0352 tribute</i>), 207	at-
stanza (<i>slixmpp.plugins.xep_0152.XEP_0152 tribute</i>), 174	at-	stanza (<i>slixmpp.plugins.xep_0353.XEP_0353 tribute</i>), 208	at-
stanza (<i>slixmpp.plugins.xep_0153.XEP_0153 tribute</i>), 175	at-	stanza (<i>slixmpp.plugins.xep_0359.XEP_0359 tribute</i>), 209	at-
stanza (<i>slixmpp.plugins.xep_0172.XEP_0172 tribute</i>), 177	at-	stanza (<i>slixmpp.plugins.xep_0369.XEP_0369 tribute</i>), 212	at-
stanza (<i>slixmpp.plugins.xep_0184.XEP_0184 tribute</i>), 178	at-	stanza (<i>slixmpp.plugins.xep_0377.XEP_0377 tribute</i>), 215	at-
stanza (<i>slixmpp.plugins.xep_0191.XEP_0191 tribute</i>), 180	at-	stanza (<i>slixmpp.plugins.xep_0394.XEP_0394 tribute</i>), 216	at-
stanza (<i>slixmpp.plugins.xep_0196.XEP_0196 tribute</i>), 181	at-	stanza (<i>slixmpp.plugins.xep_0403.XEP_0403 tribute</i>), 218	at-
stanza (<i>slixmpp.plugins.xep_0198.XEP_0198 tribute</i>), 181	at-	stanza (<i>slixmpp.plugins.xep_0404.XEP_0404 tribute</i>), 219	at-
stanza (<i>slixmpp.plugins.xep_0199.XEP_0199 tribute</i>), 185	at-	stanza (<i>slixmpp.plugins.xep_0405.XEP_0405 tribute</i>), 220	at-
stanza (<i>slixmpp.plugins.xep_0202.XEP_0202 tribute</i>), 185	at-	stanza (<i>slixmpp.plugins.xep_0421.XEP_0421 tribute</i>), 221	at-
stanza (<i>slixmpp.plugins.xep_0203.XEP_0203 tribute</i>), 187	at-	stanza (<i>slixmpp.plugins.xep_0422.XEP_0422 tribute</i>), 221	at-
stanza (<i>slixmpp.plugins.xep_0224.XEP_0224 tribute</i>), 190	at-	stanza (<i>slixmpp.plugins.xep_0424.XEP_0424 tribute</i>), 222	at-
stanza (<i>slixmpp.plugins.xep_0235.XEP_0235 tribute</i>), 191	at-	stanza (<i>slixmpp.plugins.xep_0425.XEP_0425 tribute</i>), 223	at-
stanza (<i>slixmpp.plugins.xep_0249.XEP_0249 tribute</i>), 192	at-	stanza (<i>slixmpp.plugins.xep_0428.XEP_0428 tribute</i>), 224	at-
stanza (<i>slixmpp.plugins.xep_0256.XEP_0256 tribute</i>), 193	at-	stanza (<i>slixmpp.plugins.xep_0437.XEP_0437 tribute</i>), 224	at-
stanza (<i>slixmpp.plugins.xep_0257.XEP_0257 tribute</i>), 193	at-	stanza (<i>slixmpp.plugins.xep_0439.XEP_0439 tribute</i>), 225	at-
stanza (<i>slixmpp.plugins.xep_0258.XEP_0258 tribute</i>), 194	at-	stanza (<i>slixmpp.plugins.xep_0444.XEP_0444 tribute</i>), 226	at-
stanza (<i>slixmpp.plugins.xep_0279.XEP_0279 tribute</i>), 197	at-	stanza object, 233	
stanza (<i>slixmpp.plugins.xep_0280.XEP_0280 tribute</i>), 197	at-	stanza plugin, 233	
stanza (<i>slixmpp.plugins.xep_0297.XEP_0297 tribute</i>), 198	at-	StanzaBase (<i>class in slixmpp.xmlstream.stanzabase</i>), 87	
stanza (<i>slixmpp.plugins.xep_0300.XEP_0300 tribute</i>), 199	at-	StanzaID (<i>class in slixmpp.plugins.xep_0359.stanza</i>), 209	
stanza (<i>slixmpp.plugins.xep_0308.XEP_0308 tribute</i>), 200	at-	StanzaPath (<i>class in slixmpp.xmlstream.matcher.stanzapath</i>), 92	
	at-	start_command () (<i>slixmpp.plugins.xep_0050.XEP_0050 method</i>), 128	

<code>start_stream_handler()</code>				
	<code>(slixmpp.basexmpp.BaseXMPP</code>	<code>method),</code>	<code>slixmpp.plugins.xep_0198.stanza),</code>	<code>184</code>
	<code>74</code>		<code>sub_interfaces(slixmpp.plugins.xep_0004.stanza.field.FieldOption</code>	
<code>start_stream_handler()</code>			<code>attribute),</code>	<code>101</code>
	<code>(slixmpp.componentxmpp.ComponentXMPP</code>		<code>sub_interfaces(slixmpp.plugins.xep_0004.stanza.field.FormField</code>	
	<code>method),</code>	<code>102</code>	<code>attribute),</code>	
<code>start_stream_handler()</code>			<code>sub_interfaces(slixmpp.plugins.xep_0004.stanza.form.Form</code>	
	<code>(slixmpp.xmlstream.xmlstream.XMLStream</code>		<code>attribute),</code>	<code>103</code>
	<code>method),</code>	<code>98</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCAdminItem</code>	
<code>start_tls()</code>	<code>(slixmpp.xmlstream.xmlstream.XMLStream</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCDecline</code>		
	<code>method),</code>	<code>attribute),</code>	<code>123</code>	
<code>states</code>	<code>(slixmpp.plugins.xep_0085.stanza.ChatState</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCInvite</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>123</code>	
<code>statuses</code>	<code>(slixmpp.plugins.xep_0050.stanza.Command</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCJoin</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>123</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0080.XEP_0080</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCOwnerDestroy</code>		
	<code>method),</code>	<code>attribute),</code>	<code>124</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0084.XEP_0084</code>	<code>sub_interfaces(slixmpp.plugins.xep_0045.stanza.MUCUserItem</code>		
	<code>method),</code>	<code>attribute),</code>	<code>125</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0107.XEP_0107</code>	<code>sub_interfaces(slixmpp.plugins.xep_0047.stanza.Data</code>		
	<code>method),</code>	<code>attribute),</code>	<code>125</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0108.XEP_0108</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Address</code>		
	<code>method),</code>	<code>attribute),</code>	<code>130</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0118.XEP_0118</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Agent</code>		
	<code>method),</code>	<code>attribute),</code>	<code>130</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0152.XEP_0152</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Email</code>		
	<code>method),</code>	<code>attribute),</code>	<code>132</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0172.XEP_0172</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Geo</code>		
	<code>method),</code>	<code>attribute),</code>	<code>132</code>	
<code>stop()</code>	<code>(slixmpp.plugins.xep_0196.XEP_0196</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Logo</code>		
	<code>method),</code>	<code>attribute),</code>	<code>133</code>	
<code>Store</code>	<code>(class in slixmpp.plugins.xep_0334.stanza),</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Name</code>		
	<code>206</code>	<code>attribute),</code>	<code>134</code>	
<code>store()</code>	<code>(slixmpp.plugins.xep_0222.XEP_0222</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Org</code>		
	<code>method),</code>	<code>attribute),</code>	<code>135</code>	
<code>store()</code>	<code>(slixmpp.plugins.xep_0223.XEP_0223</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Photo</code>		
	<code>method),</code>	<code>attribute),</code>	<code>135</code>	
<code>stream</code>	<code>(slixmpp.xmlstream.handler.base.BaseHandler</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Sound</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>137</code>	
<code>stream handler</code>	<code>, 233</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.Telephone</code>		
		<code>attribute),</code>	<code>137</code>	
<code>stream:[stream id]:[peer jid]</code>	<code>, 66</code>			
<code>stream_error</code>	<code>, 66</code>			
<code>stream_footer</code>	<code>(slixmpp.xmlstream.xmlstream.XMLStream</code>	<code>sub_interfaces(slixmpp.plugins.xep_0054.stanza.VCardTemp</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>138</code>	
<code>stream_header</code>	<code>(slixmpp.xmlstream.xmlstream.XMLStream</code>	<code>sub_interfaces(slixmpp.plugins.xep_0059.stanza.Set</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>140</code>	
<code>stream_id</code>	<code>(slixmpp.basexmpp.BaseXMPP</code>	<code>sub_interfaces(slixmpp.plugins.xep_0065.stanza.Socks5</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>155</code>	
<code>stream_ns</code>	<code>(slixmpp.xmlstream.xmlstream.XMLStream</code>	<code>sub_interfaces(slixmpp.plugins.xep_0066.stanza.OOB</code>		
	<code>attribute),</code>	<code>attribute),</code>	<code>156</code>	
<code>StreamHost</code>	<code>(class</code>	<code>in sub_interfaces(slixmpp.plugins.xep_0066.stanza.OOBTransfer</code>		
	<code>slixmpp.plugins.xep_0065.stanza),</code>	<code>attribute),</code>	<code>157</code>	
<code>StreamHostUsed</code>	<code>(class</code>	<code>in sub_interfaces(slixmpp.plugins.xep_0077.stanza.Register</code>		
	<code>slixmpp.plugins.xep_0065.stanza),</code>	<code>attribute),</code>	<code>158</code>	
<code>StreamManagement</code>	<code>(class</code>	<code>in sub_interfaces(slixmpp.plugins.xep_0080.stanza.Geoloc</code>		

attribute), 164
 sub_interfaces (*slixmpp.plugins.xep_0085.stanza.ChatState* interfaces (*slixmpp.plugins.xep_0425.stanza.Moderated attribute*), 224
 sub_interfaces (*slixmpp.plugins.xep_0092.stanza.Version* interfaces (*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 86
 sub_interfaces (*slixmpp.plugins.xep_0107.stanza.UserMood* interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.MethodCall attribute*), 104
 sub_interfaces (*slixmpp.plugins.xep_0108.stanza.UserActivity* interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.MethodResponse attribute*), 104
 sub_interfaces (*slixmpp.plugins.xep_0118.stanza.UserTime* interfaces (*slixmpp.plugins.xep_0009.stanza.RPC.RPCQuery attribute*), 105
 sub_interfaces (*slixmpp.plugins.xep_0122.stanza.FormValidation* (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 173
 sub_interfaces (*slixmpp.plugins.xep_0152.stanza.Address* 146
Subscribe (class in slixmpp.plugins.xep_0369.stanza),
 sub_interfaces (*slixmpp.plugins.xep_0153.stanza.VCardTempUpdate* subscribe () (*slixmpp.plugins.xep_0060.XEP_0060 attribute*), 176
 sub_interfaces (*slixmpp.plugins.xep_0184.stanza.Received* method), 142
*subscribe () (*slixmpp.plugins.xep_0437.XEP_0437 attribute*)*, 178
 sub_interfaces (*slixmpp.plugins.xep_0184.stanza.Request* method), 224
SubscribeOptions (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 179
 sub_interfaces (*slixmpp.plugins.xep_0196.stanza.UserGaming* *slixmpp.plugins.xep_0060.stanza.pubsub*), 181
147
 sub_interfaces (*slixmpp.plugins.xep_0202.stanza.EntitySubscription* (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 186
*subscribe () (*slixmpp.plugins.xep_0060.stanza.pubsub*)*, 147
 sub_interfaces (*slixmpp.plugins.xep_0235.stanza.OAuth* Subscriptions (class in *slixmpp.plugins.xep_0060.stanza.pubsub*), 191
193
 sub_interfaces (*slixmpp.plugins.xep_0257.stanza.AppendCert* *slixmpp.plugins.xep_0060.stanza.pubsub*), 193
147
 sub_interfaces (*slixmpp.plugins.xep_0257.stanza.Cert* Stanza, 233
*supports () (*slixmpp.plugins.xep_0030.XEP_0030 attribute*)*, 193
 sub_interfaces (*slixmpp.plugins.xep_0257.stanza.DisableCert* method), 113
attribute), 194
 sub_interfaces (*slixmpp.plugins.xep_0257.stanza.RevokeCert* T tag (*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 194
86
 sub_interfaces (*slixmpp.plugins.xep_0313.stanza.MAM* tag_name () (*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 201
86
 sub_interfaces (*slixmpp.plugins.xep_0313.stanza.Preferences* class method), 202
Telephone (class in slixmpp.plugins.xep_0054.stanza),
 sub_interfaces (*slixmpp.plugins.xep_0369.stanza.Join* 137
attribute), 213
terminate_command ()
 sub_interfaces (*slixmpp.plugins.xep_0369.stanza.MIX* (*slixmpp.plugins.xep_0050.XEP_0050 attribute*), 213
method), 128
 sub_interfaces (*slixmpp.plugins.xep_0369.stanza.Participant* (class in *slixmpp.plugins.xep_0377.stanza*), 214
214
TimeZone (class in slixmpp.plugins.xep_0054.stanza),
 sub_interfaces (*slixmpp.plugins.xep_0369.stanza.Setnick* 137
attribute), 214
Title (class in slixmpp.plugins.xep_0054.stanza), 137
 sub_interfaces (*slixmpp.plugins.xep_0377.stanza.Report_b64* () (in module *slixmpp.plugins.xep_0047.stanza*), 125
attribute), 215
 sub_interfaces (*slixmpp.plugins.xep_0403.stanza.MIXPresence* () (in module *slixmpp.xmlstream*), 99
attribute), 218
*true_values (*slixmpp.plugins.xep_0004.stanza.field.FormField attribute*)*, 102
attribute), 219
 sub_interfaces (*slixmpp.plugins.xep_0425.stanza.Moderate*

U

UID (*class in slixmpp.plugins.xep_0054.stanza*), 138
 Unblock (*class in slixmpp.plugins.xep_0191.stanza*), 180
 unescape () (*slixmpp.jid.JID method*), 75
 unhandled () (*slixmpp.stanza.Iq method*), 232
 unhandled () (*slixmpp.xmlstream.stanzabase.StanzaBase method*), 88
 Unsubscribe (*class in slixmpp.plugins.xep_0060.stanza.pubsub*), 147
 Unsubscribe (*class in slixmpp.plugins.xep_0369.stanza*), 214
 unsubscribe () (*slixmpp.plugins.xep_0060.XEP_0060 method*), 143
 unsubscribe () (*slixmpp.plugins.xep_0437.XEP_0437 method*), 224
 UnsupportedActions (*class in slixmpp.plugins.xep_0079.stanza*), 160
 UnsupportedConditions (*class in slixmpp.plugins.xep_0079.stanza*), 160
 update_roster () (*slixmpp.clientxmpp.ClientXMPP method*), 68
 update_subscription () (*slixmpp.plugins.xep_0369.XEP_0369 method*), 212
 UpdateSubscription (*class in slixmpp.plugins.xep_0369.stanza*), 214
 URI (*class in slixmpp.plugins.xep_0221.stanza*), 188
 URL (*class in slixmpp.plugins.xep_0054.stanza*), 138
 use_aiodns (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 98
 use_cdata (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99
 use_ipv6 (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99
 use_message_ids (*slixmpp.basexmpp.BaseXMPP attribute*), 74
 use_origin_id (*slixmpp.basexmpp.BaseXMPP attribute*), 74
 use_presence_ids (*slixmpp.basexmpp.BaseXMPP attribute*), 74
 use_proxy (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99
 use_ssl (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99
 UserActivity (*class in slixmpp.plugins.xep_0108.stanza*), 170
 UserGaming (*class in slixmpp.plugins.xep_0196.stanza*), 181
 UserMood (*class in slixmpp.plugins.xep_0107.stanza*), 169
 username () (*slixmpp.basexmpp.BaseXMPP property*), 74

UserNick (*class in slixmpp.plugins.xep_0172.stanza*), 177
 UserPreference (*class in slixmpp.plugins.xep_0404.stanza*), 219
 UserTune (*class in slixmpp.plugins.xep_0118.stanza*), 172

V

in values () (*slixmpp.xmlstream.stanzabase.ElementBase property*), 86
 VCardTemp (*class in slixmpp.plugins.xep_0054.stanza*), 138
 VCardTempUpdate (*class in slixmpp.plugins.xep_0153.stanza*), 175
 verify_signature () (*slixmpp.plugins.xep_0235.stanza OAuth method*), 191
 Version (*class in slixmpp.plugins.xep_0092.stanza*), 168
 Visible (*class in slixmpp.plugins.xep_0186.stanza*), 179

W

wait () (*slixmpp.xmlstream.handler.Waiter method*), 91
 wait_until () (*slixmpp.xmlstream.xmlstream.XMLStream method*), 99
 Waiter (*class in slixmpp.xmlstream.handler*), 91
 whitespace_keepalive (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99
 whitespace_keepalive_interval (*slixmpp.xmlstream.xmlstream.XMLStream attribute*), 99

X

XEP_0004 (*class in slixmpp.plugins.xep_0004*), 100
 XEP_0009 (*class in slixmpp.plugins.xep_0009*), 104
 XEP_0012 (*class in slixmpp.plugins.xep_0012*), 105
 XEP_0013 (*class in slixmpp.plugins.xep_0013*), 105
 XEP_0020 (*class in slixmpp.plugins.xep_0020*), 107
 XEP_0027 (*class in slixmpp.plugins.xep_0027*), 107
 XEP_0030 (*class in slixmpp.plugins.xep_0030*), 108
 XEP_0033 (*class in slixmpp.plugins.xep_0033*), 118
 XEP_0045 (*class in slixmpp.plugins.xep_0045*), 120
 XEP_0047 (*class in slixmpp.plugins.xep_0047*), 125
 XEP_0049 (*class in slixmpp.plugins.xep_0049*), 126
 XEP_0050 (*class in slixmpp.plugins.xep_0050*), 126
 XEP_0054 (*class in slixmpp.plugins.xep_0054*), 130
 XEP_0059 (*class in slixmpp.plugins.xep_0059*), 139
 XEP_0060 (*class in slixmpp.plugins.xep_0060*), 140
 XEP_0065 (*class in slixmpp.plugins.xep_0065*), 154
 XEP_0066 (*class in slixmpp.plugins.xep_0066*), 156
 XEP_0070 (*class in slixmpp.plugins.xep_0070*), 157
 XEP_0071 (*class in slixmpp.plugins.xep_0071*), 157

XEP_0077 (*class in slixmpp.plugins.xep_0077*), 158
XEP_0079 (*class in slixmpp.plugins.xep_0079*), 159
XEP_0080 (*class in slixmpp.plugins.xep_0080*), 161
XEP_0082 (*class in slixmpp.plugins.xep_0082*), 164
XEP_0084 (*class in slixmpp.plugins.xep_0084*), 164
XEP_0085 (*class in slixmpp.plugins.xep_0085*), 165
XEP_0086 (*class in slixmpp.plugins.xep_0086*), 167
XEP_0092 (*class in slixmpp.plugins.xep_0092*), 168
XEP_0106 (*class in slixmpp.plugins.xep_0106*), 169
XEP_0107 (*class in slixmpp.plugins.xep_0107*), 169
XEP_0108 (*class in slixmpp.plugins.xep_0108*), 170
XEP_0115 (*class in slixmpp.plugins.xep_0115*), 171
XEP_0118 (*class in slixmpp.plugins.xep_0118*), 171
XEP_0122 (*class in slixmpp.plugins.xep_0122*), 172
XEP_0128 (*class in slixmpp.plugins.xep_0128*), 173
XEP_0131 (*class in slixmpp.plugins.xep_0131*), 174
XEP_0133 (*class in slixmpp.plugins.xep_0133*), 174
XEP_0152 (*class in slixmpp.plugins.xep_0152*), 174
XEP_0153 (*class in slixmpp.plugins.xep_0153*), 175
XEP_0163 (*class in slixmpp.plugins.xep_0163*), 176
XEP_0172 (*class in slixmpp.plugins.xep_0172*), 177
XEP_0184 (*class in slixmpp.plugins.xep_0184*), 178
XEP_0186 (*class in slixmpp.plugins.xep_0186*), 179
XEP_0191 (*class in slixmpp.plugins.xep_0191*), 180
XEP_0196 (*class in slixmpp.plugins.xep_0196*), 180
XEP_0198 (*class in slixmpp.plugins.xep_0198*), 181
XEP_0199 (*class in slixmpp.plugins.xep_0199*), 184
XEP_0202 (*class in slixmpp.plugins.xep_0202*), 185
XEP_0203 (*class in slixmpp.plugins.xep_0203*), 187
XEP_0221 (*class in slixmpp.plugins.xep_0221*), 187
XEP_0222 (*class in slixmpp.plugins.xep_0222*), 188
XEP_0223 (*class in slixmpp.plugins.xep_0223*), 189
XEP_0224 (*class in slixmpp.plugins.xep_0224*), 190
XEP_0231 (*class in slixmpp.plugins.xep_0231*), 191
XEP_0235 (*class in slixmpp.plugins.xep_0235*), 191
XEP_0249 (*class in slixmpp.plugins.xep_0249*), 192
XEP_0256 (*class in slixmpp.plugins.xep_0256*), 193
XEP_0257 (*class in slixmpp.plugins.xep_0257*), 193
XEP_0258 (*class in slixmpp.plugins.xep_0258*), 194
XEP_0279 (*class in slixmpp.plugins.xep_0279*), 197
XEP_0280 (*class in slixmpp.plugins.xep_0280*), 197
XEP_0297 (*class in slixmpp.plugins.xep_0297*), 198
XEP_0300 (*class in slixmpp.plugins.xep_0300*), 199
XEP_0308 (*class in slixmpp.plugins.xep_0308*), 200
XEP_0313 (*class in slixmpp.plugins.xep_0313*), 200
XEP_0319 (*class in slixmpp.plugins.xep_0319*), 202
XEP_0332 (*class in slixmpp.plugins.xep_0332*), 202
XEP_0333 (*class in slixmpp.plugins.xep_0333*), 205
XEP_0334 (*class in slixmpp.plugins.xep_0334*), 206
XEP_0335 (*class in slixmpp.plugins.xep_0335*), 207
XEP_0352 (*class in slixmpp.plugins.xep_0352*), 207
XEP_0353 (*class in slixmpp.plugins.xep_0353*), 208
XEP_0359 (*class in slixmpp.plugins.xep_0359*), 209
XEP_0369 (*class in slixmpp.plugins.xep_0369*), 211
XEP_0377 (*class in slixmpp.plugins.xep_0377*), 215
XEP_0380 (*class in slixmpp.plugins.xep_0380*), 216
XEP_0394 (*class in slixmpp.plugins.xep_0394*), 216
XEP_0403 (*class in slixmpp.plugins.xep_0403*), 218
XEP_0404 (*class in slixmpp.plugins.xep_0404*), 218
XEP_0405 (*class in slixmpp.plugins.xep_0405*), 220
XEP_0421 (*class in slixmpp.plugins.xep_0421*), 221
XEP_0422 (*class in slixmpp.plugins.xep_0422*), 221
XEP_0424 (*class in slixmpp.plugins.xep_0424*), 222
XEP_0425 (*class in slixmpp.plugins.xep_0425*), 223
XEP_0428 (*class in slixmpp.plugins.xep_0428*), 224
XEP_0437 (*class in slixmpp.plugins.xep_0437*), 224
XEP_0439 (*class in slixmpp.plugins.xep_0439*), 225
XEP_0444 (*class in slixmpp.plugins.xep_0444*), 226
XHTML_IM (*class in slixmpp.plugins.xep_0071.stanza*), 157
xml (*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 87
xml_ns (*slixmpp.xmlstream.stanzabase.ElementBase attribute*), 87
XMLStream, 55, 57
XMLStream (*class in slixmpp.xmlstream.xmlstream*), 93
XMPPError, 74