PyTS3 Documentation

Release 2.0.0b2

The py-ts3 authors <see AUTHORS.txt>

Sep 08, 2018

Contents

1 Content 3

2 Indices and tables 31

3 What is PyTS3? 33

Python Module Index 35

Hint: You are currently watching the documentation for the next upcoming version of py-ts3. This version is quite stable and ready for use. One major advantage compared to v1 is support for the TS3 Client Query API and SSH.

CHAPTER 1

Content

1.1 Installation

ts3 is registered on PyPi, so you are done with:

```
$ pip3 install ts3
```

You can update the library then with:

```
$ pip3 install --upgrade ts3
```

1.2 `ts3`

1.2.1 `ts3.common`

This module contains common functions, classes and exceptions used in almost all modules of the ts3 package.

exception `ts3.common.`**`TS3Error`**

Bases: `Exception`

This is the base class for all exceptions in this package.

1.2.2 `ts3.escape`

This module contains classes and functions used to build valid query strings and to unescape responses.

Changed in version 2.0.0: The *TS3Escape* class has been replaced by the `escape()` and `unescape()` functions, because most of its methods became obsolete with the introduction of the `TS3QueryBuilder`.

`ts3.escape.`**`escape`**`(`*value*`)`

Escapes the *value* as required by the Server Query Manual:

```
>>> escape('Hello World')
'Hello\sWorld'
>>> escape('TeamSpeak ]|[ Server')
'TeamSpeak\s]\p[\sServer'
```

Seealso *unescape()*

`ts3.escape.`**`unescape`**`(`*value*`)`

Undo the escaping used for transport:

```
>>> unescape('Hello\sWorld')
'Hello World'
>>> unescape('TeamSpeak\s]\p[\sServer')
'TeamSpeak ]|[ Server'
```

1.2.3 ts3.response

This module contains the classes to parse a TeamSpeak 3 Server Query response and to structure the data.

`exception` `ts3.response.`**`TS3ParserError`**`(`*resp*, *exc=None*`)`

Bases: *ts3.common.TS3Error*, `ValueError`

Raised, if the data could not be parsed.

`resp = None`

The TS3Response object, that has thrown the exception.

`exc = None`

The original exception, if the parsing failed due to an exception like UnicodeDecodeError.

`class` `ts3.response.`**`TS3Response`**`(`*data*`)`

Bases: `object`

Parses **ONE** response and stores it's data. If you init an instance with the data of more than one response, parsing will fail.

Note, that this class is **lazy**. This means, that the response is only parsed, if you request an attribute, that requires a parsed version of the data.

For convenience, this class supports container emualtion, so these calls are equal:

```
>>> ts3resp.parsed[0]["client_nickname"] == ts3resp[0]["client_nickname"]
True
```

Parameters **`data`** (*bytes*) – The byte string received from the server.

`data`

Getter The list of lines from the original received response.

Type list of bytes

`data_bytestr`

Getter The raw response as bytestring.

Type bytes

`parsed`

Getter The parsed response as a list of dictionaries.

Type list of dictionaries [str->str]

Raises *TS3ParserError* – If the response could not be parsed.

class ts3.response.**TS3QueryResponse**(*data*)

Bases: *ts3.response.TS3Response*

The same as *TS3Response*, but the *error* attribute is public.

error

Getter A dictionary, that contains the error id and message.

Type dict

Raises *TS3ParserError* – If the response could not be parsed.

class ts3.response.**TS3Event**(*data*)

Bases: *ts3.response.TS3Response*

The same as *TS3Response*, but the *event* attribute is public.

event

Getter A dictionary with the information about the event.

Type dict

Raises *TS3ParserError* – If the response could not be parsed.

1.2.4 ts3.definitions

This module contains the definitions described in the TeamSpeak 3 Server Manual, so that the variables can be used instead of the constans to improve the readability of the code.

class ts3.definitions.**HostMessageMode**

Bases: object

NONE = 0

don't display anything

LOG = 1

display message in chatlog

MODAL = 2

display message in modal dialog

MODALQUIT = 3

display message in modal dialog and close connection

class ts3.definitions.**HostBannerMode**

Bases: object

NOADJUST = 0

do not adjust

IGNOREASPECT = 1

adjust but ignore aspect ratio (like TeamSpeak 2)

KEEPASPECT = 2

adjust and keep aspect ratio

class ts3.definitions.**Codec**
Bases: `object`

SPEEX_NARROWBAND = 0
speex narrowband (mono, 16bit, 8kHz)

SPEEX_WIDEBAND = 1
speex wideband (mono, 16bit, 16kHz)

SPEEX_ULTRAWIDEBAND = 2
speex ultra-wideband (mono, 16bit, 32kHz)

CELT_MONO = 3
celt mono (mono, 16bit, 48kHz)

class ts3.definitions.**CodecEncryptionMode**
Bases: `object`

INDIVIDUAL = 0
configure per channel

DISABLED = 1
globally disabled

ENABLED = 2
globally enabled

class ts3.definitions.**TextMessageTargetMode**
Bases: `object`

CLIENT = 1
target is a client

CHANNEL = 2
target is a channel

SERVER = 3
target is a virtual server

class ts3.definitions.**LogLevel**
Bases: `object`

ERROR = 1
everything that is really bad

WARNING = 2
everything that might be bad

DEBUG = 3
output that might help find a problem

INFO = 4
informational output

class ts3.definitions.**ReasonIdentifier**
Bases: `object`

KICK_CHANNEL = 4
kick client from channel

KICK_SERVER = 5
kick client from server

class `ts3.definitions.`**`PermissionGroupDatabaseTypes`**

Bases: `object`

`Template = 0`
template group (used for new virtual server)

`Regular = 1`
regular group (used for regular clients)

`Query = 2`
global query group (used for ServerQuery clients)

class `ts3.definitions.`**`PermissionGroupTypes`**

Bases: `object`

`ServerGroup = 0`
server group permission

`GlobalClient = 1`
client specific permission

`Channel = 2`
channel specific permission

`ChannelGroup = 3`
channel group permission

`ChannelClient = 4`
channel-client specific permission

class `ts3.definitions.`**`TokenType`**

Bases: `object`

`ServerGroup = 0`
server group token (id1={groupID} id2=0)

`ChannelGroup = 1`
channel group token (id1={groupID} id2={channelID})

1.2.5 `ts3.query`

This module contains a high-level API for the TeamSpeak 3 *Server Query* and *Client Query plugin*.

Changed in version 2.0.0: The `TS3Connection` class has been renamed to *`TS3ServerConnection`*.

New in version 2.0.0: The *`TS3ClientConnection`* class has been added.

exception `ts3.query.`**`TS3InvalidCommandError`**(*cmd*, *valid_cmds*)

Bases: *`ts3.common.TS3Error`*, `ValueError`

Raised if a `TS3QueryBuilder` is constructed with an unknown command.

Seealso *`TS3BaseConnection.COMMAND_SET`*

`cmd = None`
The unknown command.

`valid_cmds = None`
A set with all allowed (known) commands.

exception `ts3.query.`**`TS3QueryError`**(*resp*)

Bases: *`ts3.common.TS3Error`*

Raised, if the error code of the response was not 0.

resp = None
The `TS3Response` instance with the response data.

exception `ts3.query.`**`TS3TimeoutError`**
Bases: *`ts3.common.TS3Error`*, `TimeoutError`

Raised, if a response or event could not be received due to a *timeout*.

exception `ts3.query.`**`TS3TransportError`**
Bases: *`ts3.common.TS3Error`*

Raised if something goes wrong on the transport level, e.g. a connection cannot be established or has already been closed.

Seealso `TS3Transport`

class `ts3.query.`**`TS3BaseConnection`**(*uri=None*, *tp_args=None*)
Bases: `object`

The TS3 query client.

This class provides only the methods to **handle** the connection to a TeamSpeak 3 query service. For a more convenient interface, use the *`TS3ServerConnection`* or *`TS3ClientConnection`* class.

Note, that this class supports the *with* statement:

```
with TS3BaseConnection("ssh://serveradmin:Z0YxRb7u@localhost:10022") as ts3conn:
    ts3conn.exec_("use", sid=1)

# You can also use an equal try-finally construct.
ts3conn = TS3BaseConnection()
try:
    ts3conn.open_uri("telnet://serveradmin:Z0YxRb7u@localhost:10011")
    ts3conn.exec_("use", sid=1)
finally:
    ts3conn.close()
```

Warning:

- This class is **not thread safe**.
- Do **not reuse** already connected instances.

Changed in version 2.0.0: The *send()* method has been removed, use *`exec_()`*, *`query()`* instead. SSH support The *`open_uri()`* method.

GREETING_LENGTH = None
The length of the greeting. This is the number of lines returned by the query service after successful connection.

For example, the TS3 Server Query returns these lines upon connection:

```
b'TS3\n\r'
b'Welcome to the [...] on a specific command.\n\r'
```

COMMAND_SET = set()
A set with all known commands.

is_connected()

Returns True, if the client is currently connected.

Return type bool

host
The hostname of the host of the query service.

open (*host*, *port*, *timeout=None*, *protocol='telnet'*, *tp_args=None*)
Connect to the TS3 query service.

```
# Connect using telnet.
ts3conn.open("localhost", 10011)

# Connect using ssh.
ts3conn.open("localhost", 10022, protocol="ssh", tp_args={
    "username": "serveradmin", "password": "123456"
})
```

Parameters

- **host** (*str*) – The hostname
- **port** (*int*) – The listening port of the service.
- **timeout** (*int*) – If not *None*, an exception is raised if the connection cannot be established within *timeout* seconds.
- **protocol** (*str*) – The protocol to be used. The TS3 server supports *ssh* and *telnet*, while the client only supports *telnet*.
- **tp_args** (*dict*) – A dictionary with parameters that are passed to the `connect()` method of the used transport. The SSH protocol for example requires a *username* and *password*.

Raises

- ***TS3TransportError*** – If the client is already connected or the connection cannot be established.
- ***TS3TimeoutError*** – If the connection cannot be established within the specified *timeout*.

Seealso *open_uri()*

open_uri (*uri*, *timeout=None*, *tp_args=None*)
The same as *open()*, but the host, port, username, password, ... are encoded compact in a URI.

```
>>> ts3conn.open_uri("telnet://my.server.com:10011")
>>> ts3conn.open_uri("ssh://serveradmin@123456@my.server.com:10022")
```

close (*timeout=None*)
Sends the `quit` command and closes the telnet connection.

fileno ()

Returns The fileno() of the socket object used internally.

Return type int

wait_for_event (*timeout=None*)
Blocks until an event is received or the *timeout* exceeds. The next received event is returned.

A simple event loop looks like this:

```
ts3conn.query("servernotifyregister", event="server").exec()
while True:
    ts3conn.send_keepalive()
    try:
        event = ts3conn.wait_for_event(timeout=540)
    except TS3TimeoutError:
        pass
    else:
        # Handle the received event here ...
```

Parameters **timeout** (*None or float*) – The maximum number of seconds waited for the next event.

Return type *TS3Event*

Returns The next received ts3 event.

Raises

- ***TS3TimeoutError*** –
- **TS3RecvError** –

send_keepalive()

Sends an empty query to the query service in order to prevent automatic disconnect. Make sure to call it at least once in 5 minutes.

exec_(*cmd*, **options*, ***params*)

Sends a command to the TS3 server and returns the response. Check out the *query()* method if you want to make use of pipelining and more control.

```
# use sid=1
ts3conn.exec_("use", sid=1)

# clientlist -uid -away -groups
resp = ts3conn.exec_("clientlist", "uid", "away", "groups")
```

Parameters

- **cmd** (*str*) – A TS3 command
- **options** – The options of a command without a leading minus, e.g. `'uid'`, `'away'`.
- **params** – Some parameters (key, value pairs) which modify the command, e.g. `sid=1`.

Return type *TS3QueryResponse*

Returns A object which contains all information about the response.

Seealso `wait_for_resp()`

Versionadded 2.0.0

query(*cmd*, **options*, ***params*)

Note: The *query()* method is great if you want to **fetch** data from the server or want to **pipeline** parameters on the same command.

If you are only interested in getting the command executed, then you are probably better off using *exec_()*.

Returns a new *TS3QueryBuilder* object with the first pipe being initialised with the *options* and *params*:

```
# serverlist
q = ts3conn.query("serverlist")

# clientlist -uid -away -groups
q = ts3conn.query("clientlist", "uid", "away", "groups")

# clientdbfind pattern=ScP
q = ts3conn.query("clientdbfind", pattern="ScP")

# clientdbfind pattern=FPMPSC6MXqXq751dX7BKV0JniSo= -uid
q = ts3conn.query("clientdbfind", "uid", pattern="FPMPSC6MXqXq751dX7BKV0JniSo
↪")

# clientkick reasonid=5 reasonmsg=Go\saway! clid=1|clid=2|clid=3
q = ts3conn.query("clientkick", reasonid=5, reasonmsg="Go away!")\
    .pipe(clid=1).pipe(clid=2).pipe(clid=3)

# channelmove cid=16 cpid=1 order=0
q = ts3conn.query("channelmove", cid=16, cpid=1, order=0)

# sendtextmessage targetmode=2 target=12 msg=Hello\sWorld!
q = ts3conn.query("sendtextmessage", targetmode=2, target=12, msg="Hello␣
↪World!")
```

Queries are **executed** once the *fetch()*, *first()* or *all()* is invoked:

```
# Returns a TS3Response object.
resp = q.fetch()

# Returns the first item in the response or *None*.
resp = q.first()

# Returns a list with all items in the response rather
# than a TS3Response object.
resp = q.all()
```

Parameters

- **options** – All initial options in the first pipe.
- **params** – All initial parameters (key value pairs) in the first pipe.

Return type *TS3QueryBuilder*

Returns A query builder initialised with the *options* and *params*.

Versionadded 2.0.0

exec_query(*query*, *timeout=None*)

Sends the *query* to the server, waits and returns for the response.

Parameters **query** (`TS3QueryBuilder`) – The query which should be executed.

Return type *TS3QueryResponse*

Returns A object which contains all information about the response.

Seealso `wait_for_resp()`

Versionadded 2.0.0

class `ts3.query.`**`TS3ServerConnection`**(*uri=None*, *tp_args=None*)

Bases: `ts3.query.TS3BaseConnection`

Use this class to connect to a **TS3 Server**:

```
with TS3ServerConnection("localhost") as tsconn:
    ts3conn.exec_("login", client_login_name="serveradmin", client_login_password=
↪"MyStupidPassword")
    ts3conn.exec_("use")
    ts3conn.exec_("clientkick", clid=1)

    resp = ts3conn.query("serverlist").all()
```

`GREETING_LENGTH = 2`

The typical TS3 Server greeting:

b'TS3nr' b'Welcome to the [. . .] on a specific command.nr'

`COMMAND_SET = frozenset({'clientlist', 'serverdelete', 'setclientchannelgroup', 'permf`

All server query commands as returned by the *help* command, excluding *quit*. Use `close()` instead.

`open`(*host*, *port*, *timeout=None*, *protocol='telnet'*, *tp_args=None*)

Connect to the TS3 query service.

```
# Connect using telnet.
ts3conn.open("localhost", 10011)

# Connect using ssh.
ts3conn.open("localhost", 10022, protocol="ssh", tp_args={
    "username": "serveradmin", "password": "123456"
})
```

Parameters

- **`host`** (`str`) – The hostname
- **`port`** (`int`) – The listening port of the service.
- **`timeout`** (`int`) – If not *None*, an exception is raised if the connection cannot be established within *timeout* seconds.
- **`protocol`** (`str`) – The protocol to be used. The TS3 server supports *ssh* and *telnet*, while the client only supports *telnet*.
- **`tp_args`** (`dict`) – A dictionary with parameters that are passed to the `connect()` method of the used transport. The SSH protocol for example requires a *username* and *password*.

Raises

- ***`TS3TransportError`*** – If the client is already connected or the connection cannot be established.
- ***`TS3TimeoutError`*** – If the connection cannot be established within the specified *timeout*.

Seealso `open_uri()`

class ts3.query.**TS3ClientConnection**(*uri=None*, *tp_args=None*)

Bases: *ts3.query.TS3BaseConnection*

Use this class if you want to connect to a **TS3 Client**:

```
with TS3ClientConnection("localhost") as tsconn:
    ts3conn.exec_("auth", apikey="AAAA-BBBB-CCCC-DDDD-EEEE")
    ts3conn.exec_("use")
```

GREETING_LENGTH = 4

The typical TS3 Server greeting:

b'TS3 Clientnr' b'Welcome to the TeamSpeak 3 ClientQuery interface [. . .].nr' b'Use the "auth" command to authenticate yourself. [. . .].nr' b'selected schandlerid=1nr'

COMMAND_SET = frozenset({'clientlist', 'clientnotifyregister', 'setclientchannelgroup'

All client query commands as returned by the *help* command, excluding *quit*. Use `close()` instead.

1.2.6 ts3.query_builder

This module contains a flexible query builder which is modeled after the *COMMAND SYNTAX* section in the TS3 Server Query Manual.

versionadded 2.0.0

class ts3.query_builder.**TS3QueryBuilder**(*cmd*, *ts3conn=None*, *pipes=None*)

Bases: `object`

Simplifies building a valid TS3 query.

```
# When you are interested in the whole response.
resp = TS3QueryBuilder(ts3conn, "clientkick").pipe(pattern="Ben").fetch()

# When you are only interested in the first item in the response.
resp = TS3QueryBuilder(ts3conn, "serverlist").first()

# When you are only interested in the items, but not in the actual
# response object.
resp = TS3QueryBuilder(ts3conn, "serverlist").all()
```

Please note, that query builder objects are **not immutable**.

Parameters

- **cmd** (*str*) – The name of the command to execute, e.g. `"clientkick"`.
- **ts3conn** (`TS3BaseConnection`) – The TS3 connection which will be used to send the query.
- **pipes** (*list*) – A list of `(options, params)` in which options is a *set* and *params* is a *dictionary*.

Seealso *ts3.query.TS3BaseConnection.query()*, *ts3.query.TS3BaseConnection.exec_query()*

Todo What about the crazy *properties* in the documentation, what are they?

pipe(**options*, ***params*)

Starts a new pipe:

```
>>> q = TS3QueryBuilder("clientkick").pipe(clid=1).pipe(clid=2)
>>> print(q)
'clientkick clid=1 | clid=2'
```

options(**options*)

Adds the options to the last pipe:

```
>>> q = TS3QueryBuilder("clientkick").options("foo").pipe().options("bar")
>>> print(q)
'clientkick -foo | -bar'
```

You should prefer passing the options directly to `pipe()` as it is more readable.

Note: Most commands do not support pipelining options.

params(***params*)

Adds the parameters to the last pipe:

```
>>> q = TS3QueryBuilder("clientkick")\
...     .pipe().params(clid=1)\
...     .pipe().params(clid=2)
>>> print(q)
'clientkick clid=1 | clid=2'
```

You should prefer passing the options directly to `pipe()` as it is more readable.

compile()

Compiles the query into a TS3 query command and returns it:

```
# Strings are escaped automatic.
>>> q = TS3QueryBuilder("clientkick").params(reasonid=5, reasonmsg="Go away!
↪")\
...     .pipe(clid=1).pipe(clid=2)
>>> q.compile()
'clientkick reasonid=5 reasonmsg=Go\saway! | clid=1 | clid=2'

# Booleans are turned into 0 or 1.
>>> q = TS3QueryBuilder("clientupdate").params(client_input_muted=True)
>>> q.compile()
'clientupdate client_input_muted=1'
```

Return type str

Returns A valid TS3 query command string with arguments and options.

fetch()

Executes the query and returns the `TS3QueryResponse`.

Seealso `TS3BaseConnection.exec_query()`

first()

Executes the query and returns the first item in the parsed response. Use this method if you are only interested in the first item of the response.

If the response did not contain any items, then `None` is returned.

Seealso *ts3.query.TS3BaseConnection.exec_query()*, ts3.query.TS3QueryResponse.parsed

all()
Executes the query and returns the parsed response. Use this method if you are interested in the parsed response rather than the resoonse object.

Seealso *ts3.query.TS3BaseConnection.exec_query()*, ts3.query.TS3QueryResponse.parsed

1.2.7 ts3.filetransfer

This module contains an API for the TS3 file transfer interface.

exception ts3.filetransfer.**TS3FileTransferError**
Bases: *ts3.common.TS3Error*

This is the base class for all exceptions in this module.

exception ts3.filetransfer.**TS3UploadError**(*send_size*, *err=None*)
Bases: *ts3.filetransfer.TS3FileTransferError*

Is raised, when an upload fails.

send_size = None
The number of sent bytes till the error occured.

err = None
A string describing the condition which caused the exception more precise.

exception ts3.filetransfer.**TS3DownloadError**(*read_size*, *err=None*)
Bases: *ts3.filetransfer.TS3FileTransferError*

Is raised, when a download fails.

read_size = None
The number of read bytes untill the error occured.

err = None
A string describing the condition which caused the exception more precise.

class ts3.filetransfer.**TS3FileTransfer**(*ts3conn*)
Bases: object

A high-level TS3 file transfer handler.

The recommended methods to download or upload a file are:

- *init_download()*
- *init_upload()*

You can either use the low-level class methods, e.g. *download_by_resp()* or the high-level ones like *init_download()* to handle the file transfers:

```
ts3ft = TS3FileTransfer(ts3conn)
        with open("baz.png", "rb") as file:
                ts3ft.init_upload(input_file=file, name="/baz.png", cid=2)
        with open("baz1.png", "wb") as file:
                ts3ft.init_download(output_file=file, name="/baz.png", cid=2)
```

File transports can be monitored using a *reporthook*, a function which is periodically called with the current transfer stats:

```
def reporthook(size, block_size, total_size):
    print("{}% done.".format(size/total_size))
```

classmethod get_ftid()

Returns Returns a unique id for a file transfer.

Return type int

init_download(*output_file*, *name*, *cid*, *cpw=''*, *seekpos=0*, *query_resp_hook=None*, *reporthook=None*)

This is the recommended method to download a file from a TS3 server.

name, **cid**, **cpw** and **seekpos** are the parameters for the TS3 query command **ftinitdownload**. The parameter **clientftid** is automatically created and unique for the whole runtime of the programm.

query_resp_hook, if provided, is called, when the response of the ftinitdownload query has been received. Its single parameter is the the response of the query.

For downloading the file from the server, `download()` is called. So take a look a this method for further information.

Seealso The TS3 *ftinitdownload* command

classmethod download_by_resp(*output_file*, *ftinitdownload_resp*, *seekpos=0*, *reporthook=None*, *fallbackhost=None*)

Kicks off a file download by using a query response to a *ftinitdownload* command.

This is *almost* a shortcut for:

```
>>> TS3FileTransfer.download(
...     output_file = file,
...     adr = (resp[0]["ip"], int(resp[0]["port"])),
...     ftkey = resp[0]["ftkey"],
...     seekpos = seekpos,
...     total_size = resp[0]["size"],
...     reporthook = reporthook
...     )
```

Note, that the value of `resp[0]["ip"]` is a csv list and needs to be parsed.

Seealso `download()`

classmethod download(*output_file*, *adr*, *ftkey*, *seekpos=0*, *total_size=0*, *reporthook=None*)

Downloads a file from a TS3 server in the file **output_file**. The TS3 file transfer interface is specified with the address tuple **adr** and the download with the file transfer key **ftkey**.

If **seekpos** and the total **size** are provided, the **reporthook** function (`lambda read_size, block_size, total_size: None`) is called each time a new data block has been received.

If you provide **seekpos** and **total_size**, this method will check, if the download is complete and raise a `TS3DownloadError` if not.

Note, that if **total_size** is 0 or less, each download will be considered as complete.

If no error is raised, the number of read bytes is returned.

Returns The number of received bytes.

Return type int

Raises `TS3DownloadError` – If the download is incomplete or a socket error occured.

init_upload(*input_file*, *name*, *cid*, *cpw=''*, *overwrite=True*, *resume=False*, *query_resp_hook=None*, *reporthook=None*)

This is the recommended method to upload a file to a TS3 server.

name, **cid**, **cpw**, **overwrite** and **resume** are the parameters for the TS3 query command **ftinitdownload**. The parameter **clientftid** is automatically created and unique for the whole runtime of the programm and the value of **size** is retrieved by the size of the **input_file**.

query_resp_hook, if provided, is called, when the response of the ftinitupload query has been received. Its single parameter is the the response of the query.

For uploading the file to the server `upload()` is called. So take a look at this method for further information.

Seealso The TS3 *ftinitdownload* command

classmethod upload_by_resp(*input_file*, *ftinitupload_resp*, *reporthook=None*, *fallbackhost=None*)

This is *almost* a shortcut for:

```
>>> TS3FileTransfer.upload(
...     input_file = file,
...     adr = (resp[0]["ip"], int(resp[0]["port"])),
...     ftkey = resp[0]["ftkey"],
...     seekpos = resp[0]["seekpos"],
...     reporthook = reporthook
...    )
```

Note, that the value of `resp[0]["ip"]` is a csv list and needs to be parsed.

Seealso `upload()`

classmethod upload(*input_file*, *adr*, *ftkey*, *seekpos=0*, *reporthook=None*)

Uploads the data in the file **input_file** to the TS3 server listening at the address **adr**. **ftkey** is used to authenticate the file transfer.

When the upload begins, the *get pointer* of the **input_file** is set to seekpos.

If the **reporthook** function (`lambda send_size, block_size, total_size`) is provided, it is called each time a data block has been successfully transfered.

Raises ***TS3UploadError*** – If the upload is incomplete or a socket error occured.

This package contains a Python API for the:

- TeamSpeak 3 Server Query,
- TeamSpeak 3 Client Query,
- TeamSpeak 3 Filetransfer Interface,
- and TeamSpeak 3 Query Events.

1.3 Examples

1.3.1 Endless poke

Download: `endless_poke.py`

```python
#!/usr/bin/env python3

import time
import ts3

# Telnet or SSH ?
URI = "ssh://serveradmin:Z0YxRb7u@localhost:10022"
URI = "telnet://serveradmin:Z0YxRb7u@localhost:10011"

SID = 1


def endless_poke(ts3conn, nickname, msg=None, num=100, delay=1):
    """
    Pokes all clients where *nickname* matches *num* times with the message
    *msg*. Sleeping *delay* seconds between the single pokes. If *num* is -1,
    the client is poked forever.
    """
    if msg is None:
        msg = "Stop annoying me!"

    # Get the client ids
    clients = ts3conn.query("clientfind", pattern=nickname).all()
    clients = [client["clid"] for client in clients]

    # Break, if there's no client.
    if not clients:
        return None

    # Poke them
    i = 0
    while num == -1 or i < num:
        for clid in clients:
            ts3conn.exec_("clientpoke", msg=msg, clid=clid)
        time.sleep(delay)
    return None


if __name__ == "__main__":
    with ts3.query.TS3ServerConnection(URI) as ts3conn:
        ts3conn.exec_("use", sid=SID)
        endless_poke(ts3conn, "Ben", delay=0.25)
```

1.3.2 Hello Bot

Download: `hello_bot.py`

```python
#!/usr/bin/env python3

import time
import ts3

# Telnet or SSH ?
URI = "ssh://serveradmin:Z0YxRb7u@localhost:10022"
URI = "telnet://serveradmin:Z0YxRb7u@localhost:10011"
```

(continues on next page)

(continued from previous page)

```
SID = 1


def hello_bot(ts3conn, msg=None):
    """
    Waits for new clients and says hello to them, when they join the server.
    """
    if msg is None:
        msg = "Hello :)"

    # Register for the event.
    ts3conn.exec_("servernotifyregister", event="server")

    while True:
        ts3conn.send_keepalive()

        try:
            # This method blocks, but we must sent the keepalive message at
            # least once in 10 minutes. So we set the timeout parameter to
            # 1 minutes, just to be ultra safe.
            event = ts3conn.wait_for_event(timeout=60)
        except ts3.query.TS3TimeoutError:
            pass
        else:
            # Greet new clients.
            if event[0]["reasonid"] == "0":
                print("Client '{}' connected.".format(event[0]["client_nickname"]))
                ts3conn.exec_("clientpoke", clid=event[0]["clid"], msg=msg)
    return None


if __name__ == "__main__":
    with ts3.query.TS3ServerConnection(URI) as ts3conn:
        ts3conn.exec_("use", sid=SID)
        hello_bot(ts3conn)
```

1.3.3 Mute/Unmute

Download: `hello_bot.py`

```
#!/usr/bin/env python3

import time
import ts3


APIKEY = "C2OL-77SJ-M45X-BZ6E-1PBJ-FE2M"
URI = "telnet://localhost:25639"


def mute_unmute(ts3conn):
    """Mutes the client for 10 seconds using the TS3 Client Query API."""
    ts3conn.exec_("clientupdate", client_input_muted=True, client_output_muted=True)
    time.sleep(10)
    ts3conn.exec_("clientupdate", client_input_muted=False, client_output_muted=False)
```

(continues on next page)

(continued from previous page)

```python
    return None


if __name__ == "__main__":
    with ts3.query.TS3ClientConnection(URI) as ts3conn:
        ts3conn.exec_("auth", apikey=APIKEY)
        ts3conn.exec_("use")
        mute_unmute(ts3conn)
```

1.3.4 Upload / Download

Download: `upload_download.py`

```python
#!/usr/bin/env python3

"""
This script uploads an XKCD comic to the default channel and downloads it again.
"""

import time
import webbrowser
import ts3

# Telnet or SSH ?
URI = "telnet://serveradmin:Z0YxRb7u@localhost:10011"
URI = "ssh://serveradmin:Z0YxRb7u@localhost:10022"

SID = 1


with ts3.query.TS3ServerConnection(URI) as ts3conn:
    ts3conn.exec_("use", sid=SID)

    # Get the default channel.
    resp = ts3conn.query("channellist").options("flags").all()
    cid = [item["cid"] for item in resp if item["channel_flag_default"] == "1"]
    cid = cid[0]

    # Use the convenient high-level API provided by py-ts3.
    ts3ft = ts3.filetransfer.TS3FileTransfer(ts3conn)

    # Upload the comic.
    with open("./xkcd_python.png", "rb") as file:
        ts3ft.init_upload(input_file=file, name="/comic.png", cid=cid)
    print("upload complete.")

    # Download the comic.
    with open("./xkcd_python_(copy).png", "wb") as file:
        ts3ft.init_download(output_file=file, name="/comic.png", cid=cid)
    print("download complete.")

    # Display the downloaded file.
    webbrowser.open("./xkcd_python_(copy).png")
```

1.3.5 Viewer

Download: `viewer.py`

```
#!/usr/bin/env python3

import ts3


# Telnet or SSH ?
URI = "ssh://serveradmin:Z0YxRb7u@localhost:10022"
URI = "telnet://serveradmin:Z0YxRb7u@localhost:10011"

SID = 1


__all__ = ["ChannelTreeNode",
           "view"]


class ChannelTreeNode(object):
    """
    Represents a channel or the virtual server in the channel tree of a virtual
    server. Note, that this is a recursive data structure.

    Common
    ------

    self.childs = List with the child *Channels*.

    self.root = The *Channel* object, that is the root of the whole channel
                tree.

    Channel
    -------

    Represents a real channel.

    self.info =  Dictionary with all informations about the channel obtained by
                 ts3conn.channelinfo

    self.parent = The parent channel, represented by another *Channel* object.

    self.clients = List with dictionaries, that contains informations about the
                   clients in this channel.

    Root Channel
    ------------

    Represents the virtual server itself.

    self.info = Dictionary with all informations about the virtual server
                obtained by ts3conn.serverinfo

    self.parent = None

    self.clients = None
```

(continues on next page)

(continued from previous page)

```
    Usage
    -----

    >>> tree = ChannelTreeNode.build_tree(ts3conn, sid=1)

    Todo
    ----

    * It's not sure, that the tree is always correct sorted.
    """

    def __init__(self, info, parent, root, clients=None):
        """
        Inits a new channel node.

        If root is None, root is set to *self*.
        """
        self.info = info
        self.childs = list()

        # Init a root channel
        if root is None:
            self.parent = None
            self.clients = None
            self.root = self

        # Init a real channel
        else:
            self.parent = parent
            self.root = root
            self.clients = clients if clients is not None else list()
        return None

    @classmethod
    def init_root(cls, info):
        """
        Creates a the root node of a channel tree.
        """
        return cls(info, None, None, None)

    def is_root(self):
        """
        Returns true, if this node is the root of a channel tree (the virtual
        server).
        """
        return self.parent is None

    def is_channel(self):
        """
        Returns true, if this node represents a real channel.
        """
        return self.parent is not None

    @classmethod
    def build_tree(cls, ts3conn, sid):
        """
        Returns the channel tree from the virtual server identified with
```

(continues on next page)

(continued from previous page)

```
        *sid*, using the *TS3Connection* ts3conn.
        """
        ts3conn.exec_("use", sid=sid, virtual=True)

        serverinfo = ts3conn.query("serverinfo").first()
        channellist = ts3conn.query("channellist").all()
        clientlist = ts3conn.query("clientlist").all()

        # channel id -> clients
        clientlist = {cid: [client for client in clientlist \
                            if client["cid"] == cid]
                      for cid in map(lambda e: e["cid"], channellist)}

        root = cls.init_root(serverinfo)
        for channel in channellist:
            channelinfo = ts3conn.query("channelinfo", cid=channel["cid"]).first()

            # This makes sure, that *cid* is in the dictionary.
            channelinfo.update(channel)

            channel = cls(
                info=channelinfo, parent=root, root=root,
                clients=clientlist[channel["cid"]])
            root.insert(channel)
        return root

    def insert(self, channel):
        """
        Inserts the channel in the tree.
        """
        self.root._insert(channel)
        return None

    def _insert(self, channel):
        """
        Inserts the channel recursivly in the channel tree.
        Returns true, if the tree has been inserted.
        """
        # We assumed on previous insertions, that a channel is a direct child
        # of the root, if we could not find the parent. Correct this, if ctree
        # is the parent from one of these orpheans.
        if self.is_root():
            i = 0
            while i < len(self.childs):
                child = self.childs[i]
                if channel.info["cid"] == child.info["pid"]:
                    channel.childs.append(child)
                    self.childs.pop(i)
                else:
                    i += 1

        # This is not the root and the channel is a direct child of this one.
        elif channel.info["pid"] == self.info["cid"]:
            self.childs.append(channel)
            return True

        # Try to insert the channel recursive.
```

(continues on next page)

(continued from previous page)

```
        for child in self.childs:
            if child._insert(channel):
                return True

        # If we could not find a parent in the whole tree, assume, that the
        # channel is a child of the root.
        if self.is_root():
            self.childs.append(channel)
        return False

    def print(self, indent=0):
        """
        Prints the channel and it's subchannels recursive. If restore_order is
        true, the child channels will be sorted before printing them.
        """
        if self.is_root():
            print(" "*(indent*3) + "|-", self.info["virtualserver_name"])
        else:
            print(" "*(indent*3) + "|-", self.info["channel_name"])
            for client in self.clients:
                # Ignore query clients
                if client["client_type"] == "1":
                    continue
                print(" "*(indent*3+3) + "->", client["client_nickname"])

        for child in self.childs:
            child.print(indent=indent + 1)
        return None


def view(ts3conn, sid=1):
    """
    Prints the channel tree of the virtual server, including all clients.
    """
    tree = ChannelTreeNode.build_tree(ts3conn, sid)
    tree.print()
    return None


if __name__ == "__main__":
    with ts3.query.TS3ServerConnection(URI) as ts3conn:
        ts3conn.exec_("use", sid=SID)
        view(ts3conn, sid=1)
```

1.3.6 Whirlpool

Download: `whirlpool.py`

```
#!/usr/bin/env python3

import time
import random
import ts3
from ts3.definitions import TextMessageTargetMode
```

(continues on next page)

(continued from previous page)

```
# Telnet or SSH ?
URI = "ssh://serveradmin:Z0YxRb7u@localhost:10022"
URI = "telnet://serveradmin:Z0YxRb7u@localhost:10011"

SID = 1


def whirlpool(ts3conn, duration=10, relax_time=0.5):
    """
    Moves all clients randomly in other channels for *duration* seconds.
    After the whirpool event, all clients will be in the same channel as
    before. Between the whirlpool cycles, the programm will sleep for
    *relax_time* seconds.
    """
    # Countdown till whirlpool
    for i in range(5, 0, -1):
        ts3conn.exec_(
            "sendtextmessage", targetmode=TextMessageTargetMode.SERVER,
            target=0, msg="Whirpool in {}s".format(i)
        )
        time.sleep(1)

    # Fetch the ids of all channels.
    channels = ts3conn.query("channellist").all()
    cids = [channel["cid"] for channel in channels]

    # Fetch the ids of all clients and ignore query clients.
    clients = ts3conn.query("clientlist").all()
    clids = [client["clid"] for client in clients if client["client_type"] != "1"]

    # Whirpool with one channel or no users is boring.
    if len(cids) == 1 or not clids:
        return None

    # Keep track of the current positions,
    # so that we can move all clients back when the whirpool stops.
    old_pos = {client["clid"]: client["cid"] for client in clients}

    # We need this try-final construct to make sure, that all
    # clients will be in the same channel at the end of the
    # whirlpool as to the beginning.
    try:
        end_time = time.time() + duration
        while end_time > time.time():

            # Move clients randomly around and ignore
            # 'already member of channel' errors.
            for clid in clids:
                try:
                    ts3conn.exec_("clientmove", clid=clid, cid=random.choice(cids))
                except ts3.query.TS3QueryError as err:
                    if err.resp.error["id"] != "770":
                        raise

            time.sleep(relax_time)
    finally:
        # Move all clients back, this time using *no* pipelining.
```

(continues on next page)

(continued from previous page)

```
        for clid in clids:
            try:
                ts3conn.exec_("clientmove", clid=clid, cid=old_pos[clid])
            except ts3.query.TS3QueryError as err:
                if err.resp.error["id"] != "770":
                    raise
    return None


if __name__ == "__main__":
    with ts3.query.TS3ServerConnection(URI) as ts3conn:
        ts3conn.exec_("use", sid=SID)
        whirlpool(ts3conn)
```

1.4 Changelog

- **2.0.0b2**
  - **added** SSH support (issue 70)
  - **added** `TS3BaseConnection.open_uri()`
  - **changed** The constructor of `TS3BaseConnection` accepts now a URI instead of a host and port to keep things simple, especially with the new SSH parameters.
  - **fixed** timeout handling
  - **fixed** error propagation

  Update Guide

```
# Old code
with TS3ServerConnection("localhost") as ts3conn:
    pass

# New code (1)
with TS3ServerConnection("telnet://localhost:10011") as ts3conn:
    pass

# New code (2)
with TS3ServerConnection("ssh://serveradmin:abc123@localhost:10011") as ts3conn:
    pass
```

- **2.0.0b1**
  - **added** Support for the TS3 Client Query API (issue 48)
  - **renamed** `TS3Connection` to `TS3ServerConnection`
  - **added** `TS3ClientConnection`
  - **removed** the monstrous `commands` module, use `TS3QueryBuilder` instead.
  - **removed** the `TS3Escape` class, use the `TS3QueryBuilder` and the `escape()` and `unespace()` functions instead.

  Version 2.0.0 introduces support for the client query API and pipelining query commands. This come at the costs and benefits of having a new query API.

Update Guide

```
# Old code
ts3conn.login(client_login_name="serveradmin", client_login_password="abc")
ts3conn.clientlist(away=True, uid=True)
ts3conn.clientkick(reasonmsg="Haha.", clid=42)

# New code
ts3conn.exec_("login", client_login_name="serveradmin", client_login_password="abc
↪")
ts3conn.exec_("clientlist", "away", "uid")
ts3conn.exec_("clientkick", reasonmsg="Haha", clid=42)

query = ts3conn.query("clientkick", reasonmsg="Haha").pipe(clid=42).pipe(clid=43)
resp = query.fetch()
```

In short:

1. The **command** is the first parameter of *exec_()*
2. The **options** are simple string arguments after the command.
3. The **parameters** are given as keyworkd arguments.

Update or not?

Version 1.0.0 is quite stable. If you don't need the client query API or support for pipelining, then there is no reason to update, but you should fix the version in your *requirements.txt* file.

If you start a new project, use version 2.0.0. It has only a slightly different API but offers more features, while keeping the readability.

- **1.0.4**
  - **added** fallbackhost parameter to some TS3FileTransfer methods
  - **fixed** UnicodeDecodeError caused by Android clients

    https://github.com/benediktschmitt/py-ts3/issues/34
- **1.0.0**

  All threads have been removed and the event handling has been reworked. Please take a look at the examples and the GitHub README for the new event queue.
  - **removed** *TS3ResponseRecvError*

    Use the *TS3TimeoutError* and *TS3RecvError* exceptions now.
  - **added** *TS3TimeoutError* exception
  - **added** *TS3RecvError* exception
  - **removed** *TS3BaseConnection.keepalive()*

    This method has been removed, because of the bad use of threads. You are now responsible to sent the *keepalive* message by calling *TS3BaseConnection.send_keepalive()* at least once in 10 minutes.
  - **added** *TS3BaseConnection.send_keepalive()*
  - **removed** *TS3BaseConnection.on_event()*

    use the new *TS3BaseConnection.wait_for_event()* now.
  - **removed** *TS3BaseConnection.wait_for_resp()*

    This method is an inplementation detail.

- **removed** *TS3BaseConnection.stop_recv()*

  This method is no longer needed.
- **removed** *TS3BaseConnection.recv_in_thread()*

  This method is no longer needed.
- **removed** *TS3BaseConnection.last_resp*

1.5 Frequently Asked Questions (FAQ)

If you need help or you think you found a bug, please take also a look at the GitHub issue page. If you did not found a solution for your problem, do not hesitate to open a new issue with the corresponding label (e.g. `help wanted`, `bug`, ...).

Please take also a look at *Contribute*.

1.5.1 Unexpected disconnects

anti-flood

Check the **anti-flood** settings of your TS3 server. Per default, the server limits the number of queries a host can send per minute. Take a look at the TS3 query manual to get to know how you can increase this limit or simply add the host, you are running the Python script from, to the query whitelist of your TS3 server:

```
$ # In your TS3 server folder:
$ echo "192.168.178.42" >> query_ip_whitelist.txt
```

max-idle-time

The ts3 server closes idle connections after 10 minutes automatically. You can use the `send_keepalive()` to sent an empty query to the server and thus avoid automatic disconnect. Make sure to call it at least once in 10 minutes.

1.6 Contribute

This project needs your help! Please help to improve this application and fork the repository on GitHub.

1.6.1 Bug reports

When you found a bug, please create a bug report on GitHub/Issues.

If you know how to fix the bug, you're welcome to send a *pull request*.

1.6.2 Code / Enhancements

If you want to contribute to the code or you have suggestions how we could improve the code, then tell me about it.

1.6.3 Spelling Mistakes

I guess this documentation and the source code contains a lot of spelling mistakes. Please help to reduce them.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

What is PyTS3?

It’s a small package that contains a Python 3.2+ API for

- **TS3 Server Query Events**,
- the **TS3 Server Queries API**
- the **TS3 File Transfer Interface**,
- the **TS3 Client Query API**.

Python Module Index

t

ts3, 3
ts3.common, 3
ts3.definitions, 5
ts3.escape, 3
ts3.filetransfer, 15
ts3.query, 7
ts3.query_builder, 13
ts3.response, 4

Index

A

all() (ts3.query_builder.TS3QueryBuilder method), 15

C

CELT_MONO (ts3.definitions.Codec attribute), 6
Channel (ts3.definitions.PermissionGroupTypes attribute), 7
CHANNEL (ts3.definitions.TextMessageTargetMode attribute), 6
ChannelClient (ts3.definitions.PermissionGroupTypes attribute), 7
ChannelGroup (ts3.definitions.PermissionGroupTypes attribute), 7
ChannelGroup (ts3.definitions.TokenType attribute), 7
CLIENT (ts3.definitions.TextMessageTargetMode attribute), 6
close() (ts3.query.TS3BaseConnection method), 9
cmd (ts3.query.TS3InvalidCommandError attribute), 7
Codec (class in ts3.definitions), 5
CodecEncryptionMode (class in ts3.definitions), 6
COMMAND_SET (ts3.query.TS3BaseConnection attribute), 8
COMMAND_SET (ts3.query.TS3ClientConnection attribute), 13
COMMAND_SET (ts3.query.TS3ServerConnection attribute), 12
compile() (ts3.query_builder.TS3QueryBuilder method), 14

D

data (ts3.response.TS3Response attribute), 4
data_bytestr (ts3.response.TS3Response attribute), 4
DEBUG (ts3.definitions.LogLevel attribute), 6
DISABLED (ts3.definitions.CodecEncryptionMode attribute), 6
download() (ts3.filetransfer.TS3FileTransfer class method), 16
download_by_resp() (ts3.filetransfer.TS3FileTransfer class method), 16

E

ENABLED (ts3.definitions.CodecEncryptionMode attribute), 6
err (ts3.filetransfer.TS3DownloadError attribute), 15
err (ts3.filetransfer.TS3UploadError attribute), 15
ERROR (ts3.definitions.LogLevel attribute), 6
error (ts3.response.TS3QueryResponse attribute), 5
escape() (in module ts3.escape), 3
event (ts3.response.TS3Event attribute), 5
exc (ts3.response.TS3ParserError attribute), 4
exec_() (ts3.query.TS3BaseConnection method), 10
exec_query() (ts3.query.TS3BaseConnection method), 11

F

fetch() (ts3.query_builder.TS3QueryBuilder method), 14
fileno() (ts3.query.TS3BaseConnection method), 9
first() (ts3.query_builder.TS3QueryBuilder method), 14

G

get_ftid() (ts3.filetransfer.TS3FileTransfer class method), 16
GlobalClient (ts3.definitions.PermissionGroupTypes attribute), 7
GREETING_LENGTH (ts3.query.TS3BaseConnection attribute), 8
GREETING_LENGTH (ts3.query.TS3ClientConnection attribute), 13
GREETING_LENGTH (ts3.query.TS3ServerConnection attribute), 12

H

host (ts3.query.TS3BaseConnection attribute), 9
HostBannerMode (class in ts3.definitions), 5
HostMessageMode (class in ts3.definitions), 5

I

IGNOREASPECT (ts3.definitions.HostBannerMode attribute), 5

INDIVIDUAL (ts3.definitions.CodecEncryptionMode attribute), 6
INFO (ts3.definitions.LogLevel attribute), 6
init_download() (ts3.filetransfer.TS3FileTransfer method), 16
init_upload() (ts3.filetransfer.TS3FileTransfer method), 16
is_connected() (ts3.query.TS3BaseConnection method), 8

K

KEEPASPECT (ts3.definitions.HostBannerMode attribute), 5
KICK_CHANNEL (ts3.definitions.ReasonIdentifier attribute), 6
KICK_SERVER (ts3.definitions.ReasonIdentifier attribute), 6

L

LOG (ts3.definitions.HostMessageMode attribute), 5
LogLevel (class in ts3.definitions), 6

M

MODAL (ts3.definitions.HostMessageMode attribute), 5
MODALQUIT (ts3.definitions.HostMessageMode attribute), 5

N

NOADJUST (ts3.definitions.HostBannerMode attribute), 5
NONE (ts3.definitions.HostMessageMode attribute), 5

O

open() (ts3.query.TS3BaseConnection method), 9
open() (ts3.query.TS3ServerConnection method), 12
open_uri() (ts3.query.TS3BaseConnection method), 9
options() (ts3.query_builder.TS3QueryBuilder method), 14

P

params() (ts3.query_builder.TS3QueryBuilder method), 14
parsed (ts3.response.TS3Response attribute), 4
PermissionGroupDatabaseTypes (class in ts3.definitions), 6
PermissionGroupTypes (class in ts3.definitions), 7
pipe() (ts3.query_builder.TS3QueryBuilder method), 13

Q

Query (ts3.definitions.PermissionGroupDatabaseTypes attribute), 7
query() (ts3.query.TS3BaseConnection method), 10

R

read_size (ts3.filetransfer.TS3DownloadError attribute), 15
ReasonIdentifier (class in ts3.definitions), 6
Regular (ts3.definitions.PermissionGroupDatabaseTypes attribute), 7
resp (ts3.query.TS3QueryError attribute), 7
resp (ts3.response.TS3ParserError attribute), 4

S

send_keepalive() (ts3.query.TS3BaseConnection method), 10
send_size (ts3.filetransfer.TS3UploadError attribute), 15
SERVER (ts3.definitions.TextMessageTargetMode attribute), 6
ServerGroup (ts3.definitions.PermissionGroupTypes attribute), 7
ServerGroup (ts3.definitions.TokenType attribute), 7
SPEEX_NARROWBAND (ts3.definitions.Codec attribute), 6
SPEEX_ULTRAWIDEBAND (ts3.definitions.Codec attribute), 6
SPEEX_WIDEBAND (ts3.definitions.Codec attribute), 6

T

Template (ts3.definitions.PermissionGroupDatabaseTypes attribute), 7
TextMessageTargetMode (class in ts3.definitions), 6
TokenType (class in ts3.definitions), 7
ts3 (module), 3
ts3.common (module), 3
ts3.definitions (module), 5
ts3.escape (module), 3
ts3.filetransfer (module), 15
ts3.query (module), 7
ts3.query_builder (module), 13
ts3.response (module), 4
TS3BaseConnection (class in ts3.query), 8
TS3ClientConnection (class in ts3.query), 13
TS3DownloadError, 15
TS3Error, 3
TS3Event (class in ts3.response), 5
TS3FileTransfer (class in ts3.filetransfer), 15
TS3FileTransferError, 15
TS3InvalidCommandError, 7
TS3ParserError, 4
TS3QueryBuilder (class in ts3.query_builder), 13
TS3QueryError, 7
TS3QueryResponse (class in ts3.response), 5
TS3Response (class in ts3.response), 4
TS3ServerConnection (class in ts3.query), 12
TS3TimeoutError, 8
TS3TransportError, 8

TS3UploadError, 15

U

unescape() (in module ts3.escape), 4
upload() (ts3.filetransfer.TS3FileTransfer class method), 17
upload_by_resp() (ts3.filetransfer.TS3FileTransfer class method), 17

V

valid_cmds (ts3.query.TS3InvalidCommandError attribute), 7

W

wait_for_event() (ts3.query.TS3BaseConnection method), 9
WARNING (ts3.definitions.LogLevel attribute), 6