pyxs Documentation

Release 0.1

Sergei Lebedev, Fedor Gogolev

January 12, 2016

Contents

CHAPTER 1

pyxs

Pure Python bindings for communicating with XenStore. Currently two backend options are available:

- over a Unix socket with *UnixSocketConnection*;
- over XenBus with *XenBusConnection*.

Which backend is used is determined by the arguments used for *Client* initialization, for example the following code creates a *Client* instance, working over a Unix socket:

```
>>> Client(unix_socket_path="/var/run/xenstored/socket")
<pyxs.client.Client object at 0xb74103cc>
>>> Client()
<pyxs.client.Client object at 0xb74109cc>
```

Use `xen_bus_path`, if initialize a *Client* over XenBus:

```
>>> Client(xen_bus_path="/proc/xen/xenbus")
<pyxs.client.Client object at 0xb7410d2c>
```

copyright

3. 2011 by Selectel, see AUTHORS for more details.

Contents:

CHAPTER 2

API reference

2.1 pyxs.client

This module implements XenStore client, which uses multiple connection options for communication: *UnixSocketConnection* and *XenBusConnection*. Note however, that the latter one can be a bit buggy, when dealing with WATCH_EVENT packets, so using *UnixSocketConnection* is preferable.

copyright

3. 2011 by Selectel, see AUTHORS for more details.

class pyxs.client.**UnixSocketConnection**(*path=None*, *socket_timeout=None*)
XenStore connection through Unix domain socket.

Parameters

- **path** (*str*) – path to XenStore unix domain socket, if not provided explicitly is restored from process environment – similar to what libxs does.
- **socket_timeout** (*float*) – see socket.settimeout() for details.

class pyxs.client.**XenBusConnection**(*path=None*)
XenStore connection through XenBus.

Parameters **path** (*str*) – path to XenBus block device; a predefined OS-specific constant is used, if a value isn't provided explicitly.

class pyxs.client.**Client**(*unix_socket_path=None*, *socket_timeout=None*, *xen_bus_path=None*, *connection=None*, *transaction=None*)
XenStore client – <useful comment>.

Parameters

- **xen_bus_path** (*str*) – path to XenBus device, implies that *XenBusConnection* is used as a backend.
- **unix_socket_path** (*str*) – path to XenStore Unix domain socket, usually something like /var/run/xenstored/socket – implies that *UnixSocketConnection* is used as a backend.
- **socket_timeout** (*float*) – see socket.settimeout() for details.
- **transaction** (*bool*) – if True *transaction_start()* will be issued right after connection is established.

Note: *UnixSocketConnection* is used as a fallback value, if backend cannot be determined from arguments given.

Here's a quick example:

```
>>> with Client() as c:
...     c.write("/foo/bar", "baz")
...     c.read("/foo/bar")
'OK'
'baz'
```

read(**args*)
Reads data from a given path.

Parameters **path** (*str*) – a path to read from.

write(**args*)
Writes data to a given path.

Parameters

- **value** – data to write (can be of any type, but will be coerced to `bytes()` eventually).
- **path** (*str*) – a path to write to.

mkdir(**args*)
Ensures that a given path exists, by creating it and any missing parents with empty values. If *path* or any parent already exist, its value is left unchanged.

Parameters **path** (*str*) – path to directory to create.

rm(**args*)
Ensures that a given does not exist, by deleting it and all of its children. It is not an error if *path* doesn't exist, but it **is** an error if *path*'s immediate parent does not exist either.

Parameters **path** (*str*) – path to directory to remove.

directory(**args*)
Returns a list of names of the immediate children of *path*. The resulting children are each named as `<path>/<child-leaf-name>`.

Parameters **path** (*str*) – path to list.

get_perms(**args*)
Returns a list of permissions for a given *path*, see *InvalidPermission* for details on permission format.

Parameters **path** (*str*) – path to get permissions for.

set_perms(**args*)
Sets a access permissions for a given *path*, see *InvalidPermission* for details on permission format.

Parameters

- **path** (*str*) – path to set permissions for.
- **perms** (*list*) – a list of permissions to set.

watch(**args*)
Adds a watch.

When a *path* is modified (including path creation, removal, contents change or permissions change) this generates an event on the changed *path*. Changes made in transactions cause an event only if and when committed.

Parameters

- **wpath** (*str*) – path to watch.
- **token** (*str*) – watch token, returned in watch notification.

unwatch(**args*)
Removes a previously added watch.

Parameters

- **wpath** (*str*) – path to unwatch.
- **token** (*str*) – watch token, passed to `watch()`.

wait()
Waits for any of the watched paths to generate an event, which is a `(path, token)` pair, where the first element is event path, i.e. the actual path that was modified and second element is a token, passed to the `watch()`.

get_domain_path(**args*)
Returns the domain's base path, as is used for relative transactions: ex: `"/local/domain/<domid>"`. If a given *domid* doesn't exists the answer is undefined.

Parameters **domid** (*int*) – domain to get base path for.

is_domain_introduced(**args*)
Returns `True` if `xenstored` is in communication with the domain; that is when *INTRODUCE* for the domain has not yet been followed by domain destruction or explicit *RELEASE*; and `False` otherwise.

Parameters **domid** (*int*) – domain to check status for.

introduce(**args*)
Tells `xenstored` to communicate with this domain.

Parameters

- **domid** (*int*) – a real domain id, (`0` is forbidden).
- **mfn** (*long*) – address of xenstore page in *domid*.
- **eventch** (*int*) – an unbound event chanel in *domid*.

release(**args*)
Manually requests `xenstored` to disconnect from the domain.

Parameters **domid** (*int*) – domain to disconnect.

Note: `xenstored` will in any case detect domain destruction and disconnect by itself.

---

resume(**args*)
Tells `xenstored` to clear its shutdown flag for a domain. This ensures that a subsequent shutdown will fire the appropriate watches.

Parameters **domid** (*int*) – domain to resume.

set_target(**args*)
Tells `xenstored` that a domain is targetting another one, so it should let it tinker with it. This grants domain *domid* full access to paths owned by *target*. Domain *domid* also inherits all permissions granted to *target* on all other paths.

Parameters

- **domid** (*int*) – domain to set target for.
- **target** (*int*) – target domain (yours truly, Captain).

transaction_start()
Starts a new transaction and returns transaction handle, which is simply an int.

> **Warning:** Currently `xenstored` has a bug that after 2^32 transactions it will allocate id 0 for an actual transaction.

transaction_end(*commit=True*)
End a transaction currently in progress; if no transaction is running no command is sent to XenStore.

transaction()
Returns a new *Client* instance, operating within a new transaction; can only be used only when no transaction is running. Here's an example:

```
>>> with Client().transaction() as t:
...     t.do_something()
...     t.transaction_end(commit=True)
```

However, the last line is completely optional, since the default behaviour is to commit everything on context manager exit.

Raises pyxs.exceptions.PyXSError if this client is linked to and active transaction.

2.2 pyxs.helpers

Implements various helpers.

copyright

3. 2011 by Selectel, see AUTHORS for more details.

pyxs.helpers.**compile**(*term*)
Compiles a given term to a name-validator pair, where validator is a function of a single argument, capable of validating values for *name*.

Note: *reserved* values aren't compiled, since there aren't used anywhere but in the DEBUG operation, which is not a priority.

---

pyxs.helpers.**spec**(**terms*)
Decorator, which links a given spec to the wrapped function, by updating its __spec__ attribute with a list of validators for each spec term. The following symbols can be used in term definitions:

Symbol	Description
|	A NULL (zero) byte.
<foo>	A string guaranteed not to contain any NULL bytes.
<foo|>	Binary data (which may contain zero or more NULL bytes).
<foo>|*	Zero or more strings each followed by a trailing NULL.
<foo>|+	One or more strings each followed by a trailing NULL.
?	Reserved value (may not contain NULL bytes).
??	Reserved value (may contain NULL bytes).

Note: According to `docs/misc/xenstore.txt` in the current implementation reserved values are just empty strings. So for example `"\x00\x00\x00"` is a valid `??` symbol.

---

pyxs.helpers.**compose**(**fs*)
Compose any number of one-argument functions into a single one.

```
>>> f = compose(sum, lambda x: x + 10)
>>> f([1, 2, 3])
16
```

pyxs.helpers.**many**(*f*)
Convert a one-argument predicate function to a function, which takes a various number of arguments and return `True` only when predicate is truthy for each of them; otherwise `False` is returned.

```
>>> f = many(lambda x: x > 5)
>>> f([1, 5, 9])
False
>>> f([11, 15, 19])
True
```

pyxs.helpers.**many_or_none**(*f*)
Convert a one-argument predicate function to a gunction, which takes a various number of arguments and returns `True` when predicate is truty for each of them or no arguments were provided; otherwise `False` is returned.

```
>>> f = many_or_none(lambda x: x > 5)
>>> f([])
True
>>> f([11, 15, 19])
True
```

2.3 pyxs.exceptions

This module implements a number of Python exceptions used by *pyxs* classes.

copyright

3. 2011 by Selectel, see AUTHORS for more details.

exception pyxs.exceptions.**InvalidOperation**
Exception raised when *Packet* is passed an operation, which isn't listed in *Op*.

Parameters **operation** (*int*) – invalid operation value.

exception pyxs.exceptions.**InvalidPayload**
Exception raised when `Packet` is initialized with payload, which exceeds 4096 bytes restriction or contains a trailing `NULL`.

Parameters **operation** (*bytes*) – invalid payload value.

exception pyxs.exceptions.**InvalidPath**
Exception raised when a path proccessed by a comand doesn't match the following constraints:

- its length should not exceed 3072 or 2048 for absolute and relative path respectively.
- it should only consist of ASCII alphanumerics and the four punctuation characters `-/_@` – *hyphen*, *slash*, *underscore* and *atsign*.
- it shouldn't have a trailing `/`, except for the root path.

Parameters **path** (*bytes*) – invalid path value.

exception pyxs.exceptions.**InvalidTerm**
Exception raised by *compile()* when a given term is invalid, i. e. doesn't match any of the recognized forms.

Parameters **term** (*bytes*) – invalid term value.

exception pyxs.exceptions.**InvalidPermission**
Exception raised for permission which don't match the following format:

```
w<domid>        write only
r<domid>        read only
b<domid>        both read and write
n<domid>        no access
```

Parameters **perm** (*bytes*) – invalid permission value.

exception pyxs.exceptions.**ConnectionError**
Exception raised for failures during socket operations.

exception pyxs.exceptions.**UnexpectedPacket**
Exception raised when recieved packet header doesn't match the header of the packet sent, for example if outgoing packet has `op = Op.READ` the incoming packet is expected to have `op = Op.READ` as well.

2.4 pyxs._internal

A place for secret stuff, not available in the public API.

copyright

3. 2011 by Selectel, see AUTHORS for more details.

pyxs._internal.**Op = Operations(DEBUG=0, DIRECTORY=1, READ=2, GET_PERMS=3, WATCH=4, UNWATCH=5, TR**
Operations supported by XenStore.

class pyxs._internal.**Packet**
A single message to or from XenStore.

Parameters

- **op** (*int*) – an item from *Op*, representing operation, performed by this packet.
- **payload** (*bytes*) – packet payload, should be a valid ASCII-string with characters between `[0x20;0x7f]`.
- **rq_id** (*int*) – request id – hopefuly a **unique** identifier for this packet, XenStore simply echoes this value back in reponse.
- **tx_id** (*int*) – transaction id, defaults to `0` – which means no transaction is running.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p