Python IPFS API Documentation

Release 0.4.1

py-ipfs-api team

Jul 13, 2017

Contents

1 Contents 1

2 Indices and tables 35

Python Module Index 37

CHAPTER 1

Contents

Client API Reference

All commands are accessed through the `ipfsapi.Client` class.

Exceptions

The class hierachy for exceptions is:

```
Error
 +-- VersionMismatch
 +-- EncoderError
 |    +-- EncoderMissingError
 |    +-- EncodingError
 |    +-- DecodingError
 +-- CommunicationError
      +-- ProtocolError
      +-- StatusError
      +-- ErrorResponse
      +-- ConnectionError
      +-- TimeoutError
```

exception `ipfsapi.exceptions.`**`Error`**
Base class for all exceptions in this module.

exception `ipfsapi.exceptions.`**`VersionMismatch`**(*current*, *minimum*, *maximum*)
Raised when daemon version is not supported by this client version.

exception `ipfsapi.exceptions.`**`EncoderError`**(*message*, *encoder_name*)
Base class for all encoding and decoding related errors.

exception `ipfsapi.exceptions.`**`EncoderMissingError`**(*encoder_name*)
Raised when a requested encoder class does not actually exist.

exception ipfsapi.exceptions.**EncodingError**(*encoder_name*, *original*)
Raised when encoding a Python object into a byte string has failed due to some problem with the input data.

exception ipfsapi.exceptions.**DecodingError**(*encoder_name*, *original*)
Raised when decoding a byte string to a Python object has failed due to some problem with the input data.

exception ipfsapi.exceptions.**CommunicationError**(*original*, *_message=None*)
Base class for all network communication related errors.

exception ipfsapi.exceptions.**ProtocolError**(*original*, *_message=None*)
Raised when parsing the response from the daemon has failed.

This can most likely occur if the service on the remote end isn't in fact an IPFS daemon.

exception ipfsapi.exceptions.**StatusError**(*original*, *_message=None*)
Raised when the daemon responds with an error to our request.

exception ipfsapi.exceptions.**ErrorResponse**(*message*, *original*)
Raised when the daemon has responded with an error message because the requested operation could not be carried out.

exception ipfsapi.exceptions.**ConnectionError**(*original*, *_message=None*)
Raised when connecting to the service has failed on the socket layer.

exception ipfsapi.exceptions.**TimeoutError**(*original*, *_message=None*)
Raised when the daemon didn't respond in time.

The API Client

All methods accept the following parameters in their `kwargs`:

- **opts** (*dict*) – A dictonary of custom parameters to be sent with the HTTP request

ipfsapi.**connect**(*host='localhost'*, *port=5001*, *base='api/v0'*, *chunk_size=4096*, ***defaults*)
Create a new *Client* instance and connect to the daemon to validate that its version is supported.

Raises

- *VersionMismatch*
- *ErrorResponse*
- *ConnectionError*
- *ProtocolError*
- *StatusError*
- *TimeoutError*

All parameters are identical to those passed to the constructor of the *Client* class.

Returns *~ipfsapi.Client*

ipfsapi.**assert_version**(*version*, *minimum='0.4.3'*, *maximum='0.5.0'*)
Make sure that the given daemon version is supported by this client version.

Raises *VersionMismatch*

Parameters

- **version** (*str*) – The version of an IPFS daemon.
- **minimum** (*str*) – The minimal IPFS version to allow.
- **maximum** (*str*) – The maximum IPFS version to allow.

class ipfsapi.**Client**(*host='localhost'*, *port=5001*, *base='api/v0'*, *chunk_size=4096*, ***defaults*)
Bases: object

A TCP client for interacting with an IPFS daemon.

A *Client* instance will not actually establish a connection to the daemon until at least one of it's methods is called.

Parameters

- **host** (*str*) – Hostname or IP address of the computer running the `ipfs daemon` node (defaults to the local system)
- **port** (*int*) – The API port of the IPFS deamon (usually 5001)
- **base** (*str*) – Path of the deamon's API (currently always `api/v0`)
- **chunk_size** (*int*) – The size of the chunks to break uploaded files and text content into

add(*files*, *recursive=False*, *pattern='**'*, ***kwargs*)
Add a file, or directory of files to IPFS.

```
>>> with io.open('nurseryrhyme.txt', 'w', encoding='utf-8') as f:
...     numbytes = f.write('Mary had a little lamb')
>>> c.add('nurseryrhyme.txt')
{'Hash': 'QmZfF6C9j4VtoCsTp4KSrhYH47QMd3DNXVZBKaxJdhaPab',
 'Name': 'nurseryrhyme.txt'}
```

Parameters

- **files** (*str*) – A filepath to either a file or directory
- **recursive** (*bool*) – Controls if files in subdirectories are added or not
- **pattern** (*str | list*) – Single *glob* pattern or list of *glob* patterns and compiled regular expressions to match the names of the filepaths to keep

Returns **dict** (*File name and hash of the added file node*)

get(*multihash*, ***kwargs*)
Downloads a file, or directory of files from IPFS.

Files are placed in the current working directory.

Parameters **multihash** (*str*) – The path to the IPFS object(s) to be outputted

cat(*multihash*, ***kwargs*)
Retrieves the contents of a file identified by hash.

```
>>> c.cat('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
Traceback (most recent call last):
  ...
ipfsapi.exceptions.Error: this dag node is a directory
>>> c.cat('QmeKozNssnkJ4NcyRidYgDY2jfRZqVEoRGfipkgath71bX')
b'<!DOCTYPE html>\n<html>\n\n<head>\n<title>ipfs example viewer</...'
```

Parameters **multihash** (*str*) – The path to the IPFS object(s) to be retrieved

Returns **str** (*File contents*)

ls(*multihash*, ***kwargs*)
Returns a list of objects linked to by the given hash.

```
>>> c.ls('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
{'Objects': [
  {'Hash': 'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D',
   'Links': [
    {'Hash': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7dtPNFkcNMV',
     'Name': 'Makefile',          'Size': 174, 'Type': 2},
     ...
    {'Hash': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTiYwKir8eXJY',
     'Name': 'published-version', 'Size': 55,  'Type': 2}
    ]}
  ]}
```

Parameters **multihash** (*str*) – The path to the IPFS object(s) to list links from

Returns **dict** (*Directory information and contents*)

refs (*multihash*, ***kwargs*)

Returns a list of hashes of objects referenced by the given hash.

```
>>> c.refs('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
[{'Ref': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7 ... cNMV', 'Err': ''},
 ...
 {'Ref': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTi ... eXJY', 'Err': ''}]
```

Parameters **multihash** (*str*) – Path to the object(s) to list refs from

Returns *list*

refs_local (***kwargs*)

Displays the hashes of all local objects.

```
>>> c.refs_local()
[{'Ref': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7 ... cNMV', 'Err': ''},
 ...
 {'Ref': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTi ... eXJY', 'Err': ''}]
```

Returns *list*

block_stat (*multihash*, ***kwargs*)

Returns a dict with the size of the block with the given hash.

```
>>> c.block_stat('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
{'Key':  'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D',
 'Size': 258}
```

Parameters **multihash** (*str*) – The base58 multihash of an existing block to stat

Returns **dict** (*Information about the requested block*)

block_get (*multihash*, ***kwargs*)

Returns the raw contents of a block.

```
>>> c.block_get('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
b'\x121\n"\x12 \xdaW>\x14\xe5\xc1\xf6\xe4\x92\xd1 ... \n\x02\x08\x01'
```

Parameters **multihash** (*str*) – The base58 multihash of an existing block to get

Returns **str** (*Value of the requested block*)

block_put (*file*, ***kwargs*)
Stores the contents of the given file object as an IPFS block.

```
>>> c.block_put(io.BytesIO(b'Mary had a little lamb'))
    {'Key':  'QmeV6C6XVt1wf7V7as7Yak3mxPma8jzpqyhtRtCvpKcfBb',
     'Size': 22}
```

Parameters **file** (*io.RawIOBase*) – The data to be stored as an IPFS block

Returns **dict** (*Information about the new block*) – See *block_stat()*

bitswap_wantlist (*peer=None*, ***kwargs*)
Returns blocks currently on the bitswap wantlist.

```
>>> c.bitswap_wantlist()
{'Keys': [
    'QmeV6C6XVt1wf7V7as7Yak3mxPma8jzpqyhtRtCvpKcfBb',
    'QmdCWFLDXqgdWQY9kVubbEHBbkieKd3uo7MtCm7nTZZE9K',
    'QmVQ1XvYGF19X4eJqz1s7FJYJqAxFC4oqh3vWJJEXn66cp'
]}
```

Parameters **peer** (*str*) – Peer to show wantlist for.

Returns **dict** (*List of wanted blocks*)

bitswap_stat (***kwargs*)
Returns some diagnostic information from the bitswap agent.

```
>>> c.bitswap_stat()
{'BlocksReceived': 96,
 'DupBlksReceived': 73,
 'DupDataReceived': 2560601,
 'ProviderBufLen': 0,
 'Peers': [
    'QmNZFQRxt9RMNm2VVtuV2Qx7q69bcMWRVXmr5CEkJEgJJP',
    'QmNfCubGpwYZAQxX8LQDsYgB48C4GbfZHuYdexpX9mbNyT',
    'QmNfnZ8SCs3jAtNPc8kf3WJqJqSoX7wsX7VqkLdEYMao4u',
    ...
 ],
 'Wantlist': [
    'QmeV6C6XVt1wf7V7as7Yak3mxPma8jzpqyhtRtCvpKcfBb',
    'QmdCWFLDXqgdWQY9kVubbEHBbkieKd3uo7MtCm7nTZZE9K',
    'QmVQ1XvYGF19X4eJqz1s7FJYJqAxFC4oqh3vWJJEXn66cp'
 ]
}
```

Returns **dict** (*Statistics, peers and wanted blocks*)

bitswap_unwant (*key*, ***kwargs*)
Remove a given block from wantlist.

Parameters **key** (*str*) – Key to remove from wantlist.

object_data (*multihash*, ***kwargs*)
Returns the raw bytes in an IPFS object.

```
>>> c.object_data('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
b'\x08\x01'
```

Parameters **multihash** (*str*) – Key of the object to retrieve, in base58-encoded multihash format

Returns **str** (*Raw object data*)

object_new (*template=None*, ***kwargs*)
Creates a new object from an IPFS template.

By default this creates and returns a new empty merkledag node, but you may pass an optional template argument to create a preformatted node.

```
>>> c.object_new()
{'Hash': 'QmdfTbBqBPQ7VNxZEYEj14VmRuZBkqFbiwReogJgS1zR1n'}
```

Parameters **template** (*str*) – Blueprints from which to construct the new object. Possible values:

- `"unixfs-dir"`
- `None`

Returns **dict** (*Object hash*)

object_links (*multihash*, ***kwargs*)
Returns the links pointed to by the specified object.

```
>>> c.object_links('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDx ... ca7D')
{'Hash': 'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D',
 'Links': [
    {'Hash': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7dtPNFkcNMV',
     'Name': 'Makefile',          'Size': 174},
    {'Hash': 'QmeKozNssnkJ4NcyRidYgDY2jfRZqVEoRGfipkgath71bX',
     'Name': 'example',           'Size': 1474},
    {'Hash': 'QmZAL3oHMQYqsV61tGvoAVtQLs1WzRe1zkkamv9qxqnDuK',
     'Name': 'home',              'Size': 3947},
    {'Hash': 'QmZNPyKVriMsZwJSNXeQtVQSNU4v4KEKGUQaMT61LPahso',
     'Name': 'lib',               'Size': 268261},
    {'Hash': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTiYwKir8eXJY',
     'Name': 'published-version', 'Size': 55}]}
```

Parameters **multihash** (*str*) – Key of the object to retrieve, in base58-encoded multihash format

Returns **dict** (*Object hash and merkedag links*)

object_get (*multihash*, ***kwargs*)
Get and serialize the DAG node named by multihash.

```
>>> c.object_get('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
{'Data': '{',
 'Links': [
    {'Hash': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7dtPNFkcNMV',
```

```
     'Name': 'Makefile',          'Size': 174},
    {'Hash': 'QmeKozNssnkJ4NcyRidYgDY2jfRZqVEoRGfipkgath71bX',
     'Name': 'example',           'Size': 1474},
    {'Hash': 'QmZAL3oHMQYqsV61tGvoAVtQLs1WzRe1zkkamv9qxqnDuK',
     'Name': 'home',              'Size': 3947},
    {'Hash': 'QmZNPyKVriMsZwJSNXeQtVQSNU4v4KEKGUQaMT61LPahso',
     'Name': 'lib',               'Size': 268261},
    {'Hash': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTiYwKir8eXJY',
     'Name': 'published-version', 'Size': 55}]}
```

Parameters **multihash** (*str*) – Key of the object to retrieve, in base58-encoded multihash format

Returns **dict** (*Object data and links*)

object_put (*file*, ***kwargs*)

Stores input as a DAG object and returns its key.

```
>>> c.object_put(io.BytesIO(b'''
...       {
...           "Data": "another",
...           "Links": [ {
...               "Name": "some link",
...               "Hash": "QmXg9Pp2ytZ14xgmQjYEiHjVjMFXzCV ... R39V",
...               "Size": 8
...           } ]
...       }'''))
{'Hash': 'QmZZmY4KCu9r3e7M2Pcn46Fc5qbn6NpzaAGaYb22kbfTqm',
 'Links': [
    {'Hash': 'QmXg9Pp2ytZ14xgmQjYEiHjVjMFXzCVVEcRTWJBmLgR39V',
     'Size': 8, 'Name': 'some link'}
 ]
}
```

Parameters **file** (*io.RawIOBase*) – (JSON) object from which the DAG object will be created

Returns **dict** (*Hash and links of the created DAG object*) – See `object_links()`

object_stat (*multihash*, ***kwargs*)

Get stats for the DAG node named by multihash.

```
>>> c.object_stat('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
{'LinksSize': 256, 'NumLinks': 5,
 'Hash': 'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D',
 'BlockSize': 258, 'CumulativeSize': 274169, 'DataSize': 2}
```

Parameters **multihash** (*str*) – Key of the object to retrieve, in base58-encoded multihash format

Returns *dict*

object_patch_append_data (*multihash*, *new_data*, ***kwargs*)

Creates a new merkledag object based on an existing one.

The new object will have the provided data appended to it, and will thus have a new Hash.

```
>>> c.object_patch_append_data("QmZZmY ... fTqm", io.BytesIO(b"bla"))
{'Hash': 'QmR79zQQj2aDfnrNgczUhvf2qWapEfQ82YQRt3QjrbhSb2'}
```

Parameters

- **multihash** (*str*) – The hash of an ipfs object to modify
- **new_data** (*io.RawIOBase*) – The data to append to the object's data section

Returns **dict** (*Hash of new object*)

object_patch_add_link(*root*, *name*, *ref*, *create=False*, ***kwargs*)

Creates a new merkledag object based on an existing one.

The new object will have a link to the provided object.

```
>>> c.object_patch_add_link(
...     'QmR79zQQj2aDfnrNgczUhvf2qWapEfQ82YQRt3QjrbhSb2',
...     'Johnny',
...     'QmR79zQQj2aDfnrNgczUhvf2qWapEfQ82YQRt3QjrbhSb2'
... )
{'Hash': 'QmNtXbF3AjAk59gQKRgEdVabHcSsiPUnJwHnZKyj2x8Z3k'}
```

Parameters

- **root** (*str*) – IPFS hash for the object being modified
- **name** (*str*) – name for the new link
- **ref** (*str*) – IPFS hash for the object being linked to
- **create** (*bool*) – Create intermediary nodes

Returns **dict** (*Hash of new object*)

object_patch_rm_link(*root*, *link*, ***kwargs*)

Creates a new merkledag object based on an existing one.

The new object will lack a link to the specified object.

```
>>> c.object_patch_rm_link(
...     'QmNtXbF3AjAk59gQKRgEdVabHcSsiPUnJwHnZKyj2x8Z3k',
...     'Johnny'
... )
{'Hash': 'QmR79zQQj2aDfnrNgczUhvf2qWapEfQ82YQRt3QjrbhSb2'}
```

Parameters

- **root** (*str*) – IPFS hash of the object to modify
- **link** (*str*) – name of the link to remove

Returns **dict** (*Hash of new object*)

object_patch_set_data(*root*, *data*, ***kwargs*)

Creates a new merkledag object based on an existing one.

The new object will have the same links as the old object but with the provided data instead of the old object's data contents.

```
>>> c.object_patch_set_data(
...     'QmNtXbF3AjAk59gQKRgEdVabHcSsiPUnJwHnZKyj2x8Z3k',
...     io.BytesIO(b'bla')
... )
{'Hash': 'QmSw3k2qkv4ZPsbu9DVEJaTMszAQWNgM1FTFYpfZeNQWrd'}
```

Parameters

- **root** (*str*) – IPFS hash of the object to modify
- **data** (*io.RawIOBase*) – The new data to store in root

Returns **dict** (*Hash of new object*)

file_ls(*multihash*, ***kwargs*)

Lists directory contents for Unix filesystem objects.

The result contains size information. For files, the child size is the total size of the file contents. For directories, the child size is the IPFS link size.

The path can be a prefixless reference; in this case, it is assumed that it is an `/ipfs/` reference and not `/ipns/`.

```
>>> c.file_ls('QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D')
{'Arguments': {'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D':
               'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D'},
 'Objects': {
   'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D': {
     'Hash': 'QmTkzDwWqPbnAh5YiV5VwcTLnGdwSNsNTn2aDxdXBFca7D',
     'Size': 0, 'Type': 'Directory',
     'Links': [
       {'Hash': 'Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7dtPNFkcNMV',
        'Name': 'Makefile', 'Size': 163,    'Type': 'File'},
       {'Hash': 'QmeKozNssnkJ4NcyRidYgDY2jfRZqVEoRGfipkgath71bX',
        'Name': 'example',  'Size': 1463,   'Type': 'File'},
       {'Hash': 'QmZAL3oHMQYqsV61tGvoAVtQLs1WzRe1zkkamv9qxqnDuK',
        'Name': 'home',     'Size': 3947,   'Type': 'Directory'},
       {'Hash': 'QmZNPyKVriMsZwJSNXeQtVQSNU4v4KEKGUQaMT61LPahso',
        'Name': 'lib',      'Size': 268261, 'Type': 'Directory'},
       {'Hash': 'QmSY8RfVntt3VdxWppv9w5hWgNrE31uctgTiYwKir8eXJY',
        'Name': 'published-version',
        'Size': 47, 'Type': 'File'}
       ]
   }
}}
```

Parameters **multihash** (*str*) – The path to the object(s) to list links from

Returns *dict*

resolve(*name*, *recursive=False*, ***kwargs*)

Accepts an identifier and resolves it to the referenced item.

There are a number of mutable name protocols that can link among themselves and into IPNS. For example IPNS references can (currently) point at an IPFS object, and DNS links can point at other DNS links, IPNS entries, or IPFS objects. This command accepts any of these identifiers.

```
>>> c.resolve("/ipfs/QmTkzDwWqPbnAh5YiV5VwcTLnGdw ... ca7D/Makefile")
{'Path': '/ipfs/Qmd2xkBfEwEs9oMTk77A6jrsgurpF3ugXSg7dtPNFkcNMV'}
>>> c.resolve("/ipns/ipfs.io")
{'Path': '/ipfs/QmTzQ1JRkWErjk39mryYw2WVaphAZNAREyMchXzYQ7c15n'}
```

Parameters

- **name** (*str*) – The name to resolve
- **recursive** (*bool*) – Resolve until the result is an IPFS name

Returns **dict** (*IPFS path of resource*)

key_list(***kwargs*)

Returns a list of generated public keys that can be used with name_publish

```
>>> c.key_list()
[{'Name': 'self',
  'Id': 'QmQf22bZar3WKmojipms22PkXH1MZGmvsqzQtuSvQE3uhm'},
 {'Name': 'example_key_name',
  'Id': 'QmQLaT5ZrCfSkXTH6rUKtVidcxj8jrW3X2h75Lug1AV7g8'}
]
```

Returns **list** (*List of dictionaries with Names and Ids of public keys.*)

key_gen(*key_name*, *type*, *size=2048*, ***kwargs*)

Adds a new public key that can be used for name_publish.

```
>>> c.key_gen('example_key_name')
{'Name': 'example_key_name',
 'Id': 'QmQLaT5ZrCfSkXTH6rUKtVidcxj8jrW3X2h75Lug1AV7g8'}
```

Parameters

- **key_name** (*str*) – Name of the new Key to be generated. Used to reference the Keys.
- **type** (*str*) – Type of key to generate. The current possible keys types are:
  - `"rsa"`
  - `"ed25519"`
- **size** (*int*) – Bitsize of key to generate

Returns **dict** (*Key name and Key Id*)

name_publish(*ipfs_path*, *resolve=True*, *lifetime='24h'*, *ttl=None*, *key=None*, ***kwargs*)

Publishes an object to IPNS.

IPNS is a PKI namespace, where names are the hashes of public keys, and the private key enables publishing new (signed) values. In publish, the default value of *name* is your own identity public key.

```
>>> c.name_publish('/ipfs/QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZK ... GZ5d')
{'Value': '/ipfs/QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d',
 'Name': 'QmVgNoP89mzpgEAAqK8owYoDEyB97MkcGvoWZir8otE9Uc'}
```

Parameters

- **ipfs_path** (*str*) – IPFS path of the object to be published

- **resolve** (*bool*) – Resolve given path before publishing
- **lifetime** (*str*) – Time duration that the record will be valid for

  Accepts durations such as `"300s"`, `"1.5h"` or `"2h45m"`. Valid units are:

  - `"ns"`
  - `"us"` (or `"µs"`)
  - `"ms"`
  - `"s"`
  - `"m"`
  - `"h"`
- **ttl** (*int*) – Time duration this record should be cached for
- **key** (*string*) – Name of the key to be used, as listed by 'ipfs key list'.

Returns **dict** (*IPNS hash and the IPFS path it points at*)

name_resolve(*name=None*, *recursive=False*, *nocache=False*, ***kwargs*)

Gets the value currently published at an IPNS name.

IPNS is a PKI namespace, where names are the hashes of public keys, and the private key enables publishing new (signed) values. In resolve, the default value of `name` is your own identity public key.

```
>>> c.name_resolve()
{'Path': '/ipfs/QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d'}
```

Parameters

- **name** (*str*) – The IPNS name to resolve (defaults to the connected node)
- **recursive** (*bool*) – Resolve until the result is not an IPFS name (default: false)
- **nocache** (*bool*) – Do not use cached entries (default: false)

Returns **dict** (*The IPFS path the IPNS hash points at*)

dns(*domain_name*, *recursive=False*, ***kwargs*)

Resolves DNS links to the referenced object.

Multihashes are hard to remember, but domain names are usually easy to remember. To create memorable aliases for multihashes, DNS TXT records can point to other DNS links, IPFS objects, IPNS keys, etc. This command resolves those links to the referenced object.

For example, with this DNS TXT record:

```
>>> import dns.resolver
>>> a = dns.resolver.query("ipfs.io", "TXT")
>>> a.response.answer[0].items[0].to_text()
'"dnslink=/ipfs/QmTzQ1JRkWErjk39mryYw2WVaphAZNAREyMchXzYQ7c15n"'
```

The resolver will give:

```
>>> c.dns("ipfs.io")
{'Path': '/ipfs/QmTzQ1JRkWErjk39mryYw2WVaphAZNAREyMchXzYQ7c15n'}
```

Parameters

- **domain_name** (*str*) – The domain-name name to resolve
- **recursive** (*bool*) – Resolve until the name is not a DNS link

Returns **dict** (*Resource were a DNS entry points to*)

pin_add(*path*, **paths*, ***kwargs*)
Pins objects to local storage.

Stores an IPFS object(s) from a given path locally to disk.

```
>>> c.pin_add("QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d")
{'Pins': ['QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d']}
```

Parameters

- **path** (*str*) – Path to object(s) to be pinned
- **recursive** (*bool*) – Recursively unpin the object linked to by the specified object(s)

Returns **dict** (*List of IPFS objects that have been pinned*)

pin_rm(*path*, **paths*, ***kwargs*)
Removes a pinned object from local storage.

Removes the pin from the given object allowing it to be garbage collected if needed.

```
>>> c.pin_rm('QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d')
{'Pins': ['QmfZY61ukoQuCX8e5Pt7v8pRfhkyxwZKZMTodAtmvyGZ5d']}
```

Parameters

- **path** (*str*) – Path to object(s) to be unpinned
- **recursive** (*bool*) – Recursively unpin the object linked to by the specified object(s)

Returns **dict** (*List of IPFS objects that have been unpinned*)

pin_ls(*type='all'*, ***kwargs*)
Lists objects pinned to local storage.

By default, all pinned objects are returned, but the `type` flag or arguments can restrict that to a specific pin type or to some specific objects respectively.

```
>>> c.pin_ls()
{'Keys': {
    'QmNNPMA1eGUbKxeph6yqV8ZmRkdVat ... YMuz': {'Type': 'recursive'},
    'QmNPZUCeSN5458Uwny8mXSWubjjr6J ... kP5e': {'Type': 'recursive'},
    'QmNg5zWpRMxzRAVg7FTQ3tUxVbKj8E ... gHPz': {'Type': 'indirect'},
    ...
    'QmNiuVapnYCrLjxyweHeuk6Xdqfvts ... wCCe': {'Type': 'indirect'}}}
```

Parameters **type** (*"str"*) – The type of pinned keys to list. Can be:

- `"direct"`
- `"indirect"`
- `"recursive"`
- `"all"`

Returns **dict** (*Hashes of pinned IPFS objects and why they are pinned*)

repo_gc(***kwargs*)

Removes stored objects that are not pinned from the repo.

```
>>> c.repo_gc()
[{'Key': 'QmNPXDC6wTXVmZ9Uoc8X1oqxRRJr4f1sDuyQuwaHG2mpW2'},
 {'Key': 'QmNtXbF3AjAk59gQKRgEdVabHcSsiPUnJwHnZKyj2x8Z3k'},
 {'Key': 'QmRVBnxUCsD57ic5FksKYadtyUbMsyo9KYQKKELajqAp4q'},
 ...
 {'Key': 'QmYp4TeCurXrhsxnzt5wqLqqUz8ZRg5zsc7GuUrUSDtwzP'}]
```

Performs a garbage collection sweep of the local set of stored objects and remove ones that are not pinned in order to reclaim hard disk space. Returns the hashes of all collected objects.

Returns **dict** (*List of IPFS objects that have been removed*)

repo_stat(***kwargs*)

Displays the repo's status.

Returns the number of objects in the repo and the repo's size, version, and path.

```
>>> c.repo_stat()
{'NumObjects': 354,
 'RepoPath': '.../.local/share/ipfs',
 'Version': 'fs-repo@4',
 'RepoSize': 13789310}
```

Returns **dict** (*General information about the IPFS file repository*)

NumObjects	Number of objects in the local repo.
RepoPath	The path to the repo being currently used.
RepoSize	Size in bytes that the repo is currently using.
Version	The repo version.

id(*peer=None*, ***kwargs*)

Shows IPFS Node ID info.

Returns the PublicKey, ProtocolVersion, ID, AgentVersion and Addresses of the connected daemon or some other node.

```
>>> c.id()
{'ID': 'QmVgNoP89mzpgEAAqK8owYoDEyB97MkcGvoWZir8otE9Uc',
'PublicKey': 'CAASpgIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggE ... BAAE=',
'AgentVersion': 'go-libp2p/3.3.4',
'ProtocolVersion': 'ipfs/0.1.0',
'Addresses': [
    '/ip4/127.0.0.1/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8owYo ... E9Uc',
    '/ip4/10.1.0.172/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8owY ... E9Uc',
    '/ip4/172.18.0.1/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8owY ... E9Uc',
    '/ip6/::1/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8owYoDEyB97 ... E9Uc',
    '/ip6/fccc:7904:b05b:a579:957b:deef:f066:cad9/tcp/400 ... E9Uc',
    '/ip6/fd56:1966:efd8::212/tcp/4001/ipfs/QmVgNoP89mzpg ... E9Uc',
    '/ip6/fd56:1966:efd8:0:def1:34d0:773:48f/tcp/4001/ipf ... E9Uc',
    '/ip6/2001:db8:1::1/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8 ... E9Uc',
    '/ip4/77.116.233.54/tcp/4001/ipfs/QmVgNoP89mzpgEAAqK8 ... E9Uc',
    '/ip4/77.116.233.54/tcp/10842/ipfs/QmVgNoP89mzpgEAAqK ... E9Uc']}
```

Parameters **peer** (*str*) – Peer.ID of the node to look up (local node if `None`)

Returns **dict** (*Information about the IPFS node*)

bootstrap(***kwargs*)
Compatiblity alias for *bootstrap_list()*.

bootstrap_list(***kwargs*)
Returns the addresses of peers used during initial discovery of the IPFS network.

Peers are output in the format `<multiaddr>/<peerID>`.

```
>>> c.bootstrap_list()
{'Peers': [
    '/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYER ... uvuJ',
    '/ip4/104.236.176.52/tcp/4001/ipfs/QmSoLnSGccFuZQJzRa ... ca9z',
    '/ip4/104.236.179.241/tcp/4001/ipfs/QmSoLPppuBtQSGwKD ... KrGM',
    ...
    '/ip4/178.62.61.185/tcp/4001/ipfs/QmSoLMeWqB7YGVLJN3p ... QBU3']}
```

Returns **dict** (*List of known bootstrap peers*)

bootstrap_add(*peer*, **peers*, ***kwargs*)
Adds peers to the bootstrap list.

Parameters **peer** (*str*) – IPFS MultiAddr of a peer to add to the list

Returns *dict*

bootstrap_rm(*peer*, **peers*, ***kwargs*)
Removes peers from the bootstrap list.

Parameters **peer** (*str*) – IPFS MultiAddr of a peer to remove from the list

Returns *dict*

swarm_peers(***kwargs*)
Returns the addresses & IDs of currently connected peers.

```
>>> c.swarm_peers()
{'Strings': [
    '/ip4/101.201.40.124/tcp/40001/ipfs/QmZDYAhmMDtnoC6XZ ... kPZc',
    '/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYER ... uvuJ',
    '/ip4/104.223.59.174/tcp/4001/ipfs/QmeWdgoZezpdHz1PX8 ... 1jB6',
    ...
    '/ip6/fce3: ... :f140/tcp/43901/ipfs/QmSoLnSGccFuZQJzRa ... ca9z']}
```

Returns **dict** (*List of multiaddrs of currently connected peers*)

swarm_addrs(***kwargs*)
Returns the addresses of currently connected peers by peer id.

```
>>> pprint(c.swarm_addrs())
{'Addrs': {
    'QmNMVHJTSZHTWMWBbmBrQgkA1hZPWYuVJx2DpSGESWW6Kn': [
        '/ip4/10.1.0.1/tcp/4001',
        '/ip4/127.0.0.1/tcp/4001',
        '/ip4/51.254.25.16/tcp/4001',
        '/ip6/2001:41d0:b:587:3cae:6eff:fe40:94d8/tcp/4001',
        '/ip6/2001:470:7812:1045::1/tcp/4001',
```

```
        '/ip6/::1/tcp/4001',
        '/ip6/fc02:2735:e595:bb70:8ffc:5293:8af8:c4b7/tcp/4001',
        '/ip6/fd00:7374:6172:100::1/tcp/4001',
        '/ip6/fd20:f8be:a41:0:c495:aff:fe7e:44ee/tcp/4001',
        '/ip6/fd20:f8be:a41::953/tcp/4001'],
    'QmNQsK1Tnhe2Uh2t9s49MJjrz7wgPHj4VyrZzjRe8dj7KQ': [
        '/ip4/10.16.0.5/tcp/4001',
        '/ip4/127.0.0.1/tcp/4001',
        '/ip4/172.17.0.1/tcp/4001',
        '/ip4/178.62.107.36/tcp/4001',
        '/ip6/::1/tcp/4001'],
    ...
}}
```

Returns **dict** (*Multiaddrs of peers by peer id*)

swarm_connect (*address*, **addresses*, ***kwargs*)

Opens a connection to a given address.

This will open a new direct connection to a peer address. The address format is an IPFS multiaddr:

```
/ip4/104.131.131.82/tcp/4001/ipfs/
↪QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
```

```
>>> c.swarm_connect("/ip4/104.131.131.82/tcp/4001/ipfs/Qma ... uvuJ")
{'Strings': ['connect QmaCpDMGvV2BGHeYERUEnRQAwe3 ... uvuJ success']}
```

Parameters **address** (*str*) – Address of peer to connect to

Returns **dict** (*Textual connection status report*)

swarm_disconnect (*address*, **addresses*, ***kwargs*)

Closes the connection to a given address.

This will close a connection to a peer address. The address format is an IPFS multiaddr:

```
/ip4/104.131.131.82/tcp/4001/ipfs/
↪QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
```

The disconnect is not permanent; if IPFS needs to talk to that address later, it will reconnect.

```
>>> c.swarm_disconnect("/ip4/104.131.131.82/tcp/4001/ipfs/Qm ... uJ")
{'Strings': ['disconnect QmaCpDMGvV2BGHeYERUEnRQA ... uvuJ success']}
```

Parameters **address** (*str*) – Address of peer to disconnect from

Returns **dict** (*Textual connection status report*)

swarm_filters_add (*address*, **addresses*, ***kwargs*)

Adds a given multiaddr filter to the filter list.

This will add an address filter to the daemons swarm. Filters applied this way will not persist daemon reboots, to achieve that, add your filters to the configuration file.

```
>>> c.swarm_filters_add("/ip4/192.168.0.0/ipcidr/16")
{'Strings': ['/ip4/192.168.0.0/ipcidr/16']}
```

Parameters **address** (*str*) – Multiaddr to filter

Returns **dict** (*List of swarm filters added*)

swarm_filters_rm(*address*, **addresses*, ***kwargs*)
Removes a given multiaddr filter from the filter list.

This will remove an address filter from the daemons swarm. Filters removed this way will not persist daemon reboots, to achieve that, remove your filters from the configuration file.

```
>>> c.swarm_filters_rm("/ip4/192.168.0.0/ipcidr/16")
{'Strings': ['/ip4/192.168.0.0/ipcidr/16']}
```

Parameters **address** (*str*) – Multiaddr filter to remove

Returns **dict** (*List of swarm filters removed*)

dht_query(*peer_id*, **peer_ids*, ***kwargs*)
Finds the closest Peer IDs to a given Peer ID by querying the DHT.

```
>>> c.dht_query("/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDM ... uvuJ")
[{'ID': 'QmPkFbxAQ7DeKD5VGSh9HQrdS574pyNzDmxJeGrRJxoucF',
  'Extra': '', 'Type': 2, 'Responses': None},
 {'ID': 'QmR1MhHVLJSLt9ZthsNNhudb1ny1WdhY4FPW21ZYFWec4f',
  'Extra': '', 'Type': 2, 'Responses': None},
 {'ID': 'Qmcwx1K5aVme45ab6NYWb52K2TFBeABgCLccC7ntUeDsAs',
  'Extra': '', 'Type': 2, 'Responses': None},
 ...
 {'ID': 'QmYYy8L3YD1nsF4xtt4xmsc14yqvAAnKksjo3F3iZs5jPv',
  'Extra': '', 'Type': 1, 'Responses': []}]
```

Parameters **peer_id** (*str*) – The peerID to run the query against

Returns **dict** (*List of peers IDs*)

dht_findprovs(*multihash*, **multihashes*, ***kwargs*)
Finds peers in the DHT that can provide a specific value.

```
>>> c.dht_findprovs("QmNPXDC6wTXVmZ9Uoc8X1oqxRRJr4f1sDuyQu ... mpW2")
[{'ID': 'QmaxqKpiYNr62uSFBhxJAMmEMkT6dvc3oHkrZNpH2VMTLZ',
  'Extra': '', 'Type': 6, 'Responses': None},
 {'ID': 'QmaK6Aj5WXkfnWGoWq7V8pGUYzcHPZp4jKQ5JtmRvSzQGk',
  'Extra': '', 'Type': 6, 'Responses': None},
 {'ID': 'QmdUdLu8dNvr4MVW1iWXxKoQrbG6y1vAVWPdkeGK4xppds',
  'Extra': '', 'Type': 6, 'Responses': None},
 ...
 {'ID': '', 'Extra': '', 'Type': 4, 'Responses': [
    {'ID': 'QmVgNoP89mzpgEAAqK8owYoDEyB97Mk ... E9Uc', 'Addrs': None}
  ]},
 {'ID': 'QmaxqKpiYNr62uSFBhxJAMmEMkT6dvc3oHkrZNpH2VMTLZ',
  'Extra': '', 'Type': 1, 'Responses': [
    {'ID': 'QmSHXfsmN3ZduwFDjeqBn1C8b1tcLkxK6yd ... waXw', 'Addrs': [
        '/ip4/127.0.0.1/tcp/4001',
        '/ip4/172.17.0.8/tcp/4001',
        '/ip6/::1/tcp/4001',
        '/ip4/52.32.109.74/tcp/1028'
      ]}
  ]}]
```

Parameters **multihash** (*str*) – The DHT key to find providers for

Returns **dict** (*List of provider Peer IDs*)

dht_findpeer (*peer_id*, **peer_ids*, ***kwargs*)
Queries the DHT for all of the associated multiaddresses.

```
>>> c.dht_findpeer("QmaxqKpiYNr62uSFBhxJAMmEMkT6dvc3oHkrZN ... MTLZ")
[{'ID': 'QmfVGMFrwW6AV6fTWmD6eocaTybffqAvkVLXQEFrYdk6yc',
  'Extra': '', 'Type': 6, 'Responses': None},
 {'ID': 'QmTKiUdjbRjeN9yPhNhG1X38YNuBdjeiV9JXYWzCAJ4mj5',
  'Extra': '', 'Type': 6, 'Responses': None},
 {'ID': 'QmTGkgHSsULk8p3AKTAqKixxidZQXFyF7mCURcutPqrwjQ',
  'Extra': '', 'Type': 6, 'Responses': None},
 ...
 {'ID': '', 'Extra': '', 'Type': 2,
  'Responses': [
    {'ID': 'QmaxqKpiYNr62uSFBhxJAMmEMkT6dvc3oHkrZNpH2VMTLZ',
     'Addrs': [
        '/ip4/10.9.8.1/tcp/4001',
        '/ip6/::1/tcp/4001',
        '/ip4/164.132.197.107/tcp/4001',
        '/ip4/127.0.0.1/tcp/4001']}
  ]}]
```

Parameters **peer_id** (*str*) – The ID of the peer to search for

Returns **dict** (*List of multiaddrs*)

dht_get (*key*, **keys*, ***kwargs*)
Queries the DHT for its best value related to given key.

There may be several different values for a given key stored in the DHT; in this context *best* means the record that is most desirable. There is no one metric for *best*: it depends entirely on the key type. For IPNS, *best* is the record that is both valid and has the highest sequence number (freshest). Different key types may specify other rules for they consider to be the *best*.

Parameters **key** (*str*) – One or more keys whose values should be looked up

Returns *str*

dht_put (*key*, *value*, ***kwargs*)
Writes a key/value pair to the DHT.

Given a key of the form `/foo/bar` and a value of any form, this will write that value to the DHT with that key.

Keys have two parts: a keytype (foo) and the key name (bar). IPNS uses the `/ipns/` keytype, and expects the key name to be a Peer ID. IPNS entries are formatted with a special strucutre.

You may only use keytypes that are supported in your `ipfs` binary: `go-ipfs` currently only supports the `/ipns/` keytype. Unless you have a relatively deep understanding of the key's internal structure, you likely want to be using the *name_publish()* instead.

Value is arbitrary text.

```
>>> c.dht_put("QmVgNoP89mzpgEAAqK8owYoDEyB97Mkc ... E9Uc", "test123")
[{'ID': 'QmfLy2aqbhU1RqZnGQyqHSovV8tDufLUaPfN1LNtg5CvDZ',
  'Extra': '', 'Type': 5, 'Responses': None},
 {'ID': 'QmZ5qTkNvvZ5eFq9T4dcCEK7kX8L7iysYEpvQmij9vokGE',
```

```
  'Extra': '', 'Type': 5, 'Responses': None},
 {'ID': 'QmYqa6QHCbe6eKiiW6YoThU5yBy8c3eQzpiuW22SgVWSB8',
  'Extra': '', 'Type': 6, 'Responses': None},
 ...
 {'ID': 'QmP6TAKVDCziLmx9NV8QGekwtf7ZMuJnmbeHMjcfoZbRMd',
  'Extra': '', 'Type': 1, 'Responses': []}]
```

Parameters

- **key** (*str*) – A unique identifier
- **value** (*str*) – Abitrary text to associate with the input (2048 bytes or less)

Returns *list*

ping (*peer*, **peers*, ***kwargs*)

Provides round-trip latency information for the routing system.

Finds nodes via the routing system, sends pings, waits for pongs, and prints out round-trip latency information.

```
>>> c.ping("QmTzQ1JRkWErjk39mryYw2WVaphAZNAREyMchXzYQ7c15n")
[{'Success': True,  'Time': 0,
  'Text': 'Looking up peer QmTzQ1JRkWErjk39mryYw2WVaphAZN ... c15n'},
 {'Success': False, 'Time': 0,
  'Text': 'Peer lookup error: routing: not found'}]
```

Parameters

- **peer** (*str*) – ID of peer to be pinged
- **count** (*int*) – Number of ping messages to send (Default: 10)

Returns **list** (*Progress reports from the ping*)

config (*key*, *value=None*, ***kwargs*)

Controls configuration variables.

```
>>> c.config("Addresses.Gateway")
{'Key': 'Addresses.Gateway', 'Value': '/ip4/127.0.0.1/tcp/8080'}
>>> c.config("Addresses.Gateway", "/ip4/127.0.0.1/tcp/8081")
{'Key': 'Addresses.Gateway', 'Value': '/ip4/127.0.0.1/tcp/8081'}
```

Parameters

- **key** (*str*) – The key of the configuration entry (e.g. "Addresses.API")
- **value** (*dict*) – The value to set the configuration entry to

Returns **dict** (*Requested/updated key and its (new) value*)

config_show (***kwargs*)

Returns a dict containing the server's configuration.

> **Warning:** The configuration file contains private key data that must be handled with care.

```
>>> config = c.config_show()
>>> config['Addresses']
{'API': '/ip4/127.0.0.1/tcp/5001',
 'Gateway': '/ip4/127.0.0.1/tcp/8080',
 'Swarm': ['/ip4/0.0.0.0/tcp/4001', '/ip6/::/tcp/4001']},
>>> config['Discovery']
{'MDNS': {'Enabled': True, 'Interval': 10}}
```

Returns **dict** (*The entire IPFS daemon configuration*)

config_replace(**args*, ***kwargs*)
Replaces the existing config with a user-defined config.

Make sure to back up the config file first if neccessary, as this operation can't be undone.

log_level(*subsystem*, *level*, ***kwargs*)
Changes the logging output of a running daemon.

```
>>> c.log_level("path", "info")
{'Message': "Changed log level of 'path' to 'info'\n"}
```

Parameters

- **subsystem** (*str*) – The subsystem logging identifier (Use `"all"` for all subsystems)
- **level** (*str*) – The desired logging level. Must be one of:
  - `"debug"`
  - `"info"`
  - `"warning"`
  - `"error"`
  - `"fatal"`
  - `"panic"`

Returns **dict** (*Status message*)

log_ls(***kwargs*)
Lists the logging subsystems of a running daemon.

```
>>> c.log_ls()
{'Strings': [
    'github.com/ipfs/go-libp2p/p2p/host', 'net/identify',
    'merkledag', 'providers', 'routing/record', 'chunk', 'mfs',
    'ipns-repub', 'flatfs', 'ping', 'mockrouter', 'dagio',
    'cmds/files', 'blockset', 'engine', 'mocknet', 'config',
    'commands/http', 'cmd/ipfs', 'command', 'conn', 'gc',
    'peerstore', 'core', 'coreunix', 'fsrepo', 'core/server',
    'boguskey', 'github.com/ipfs/go-libp2p/p2p/host/routed',
    'diagnostics', 'namesys', 'fuse/ipfs', 'node', 'secio',
    'core/commands', 'supernode', 'mdns', 'path', 'table',
    'swarm2', 'peerqueue', 'mount', 'fuse/ipns', 'blockstore',
    'github.com/ipfs/go-libp2p/p2p/host/basic', 'lock', 'nat',
    'importer', 'corerepo', 'dht.pb', 'pin', 'bitswap_network',
    'github.com/ipfs/go-libp2p/p2p/protocol/relay', 'peer',
    'transport', 'dht', 'offlinerouting', 'tarfmt', 'eventlog',
```

```
    'ipfsaddr', 'github.com/ipfs/go-libp2p/p2p/net/swarm/addr',
    'bitswap', 'reprovider', 'supernode/proxy', 'crypto', 'tour',
    'commands/cli', 'blockservice']}
```

Returns **dict** (*List of daemon logging subsystems*)

log_tail(***kwargs*)

Reads log outputs as they are written.

This function returns a reponse object that can be iterated over by the user. The user should make sure to close the response object when they are done reading from it.

```
>>> for item in c.log_tail():
...     print(item)
...
{"event":"updatePeer","system":"dht",
 "peerID":"QmepsDPxWtLDuKvEoafkpJxGij4kMax11uTH7WnKqD25Dq",
 "session":"7770b5e0-25ec-47cd-aa64-f42e65a10023",
 "time":"2016-08-22T13:25:27.43353297Z"}
{"event":"handleAddProviderBegin","system":"dht",
 "peer":"QmepsDPxWtLDuKvEoafkpJxGij4kMax11uTH7WnKqD25Dq",
 "session":"7770b5e0-25ec-47cd-aa64-f42e65a10023",
 "time":"2016-08-22T13:25:27.433642581Z"}
{"event":"handleAddProvider","system":"dht","duration":91704,
 "key":"QmNT9Tejg6t57Vs8XM2TVJXCwevWiGsZh3kB4HQXUZRK1o",
 "peer":"QmepsDPxWtLDuKvEoafkpJxGij4kMax11uTH7WnKqD25Dq",
 "session":"7770b5e0-25ec-47cd-aa64-f42e65a10023",
 "time":"2016-08-22T13:25:27.433747513Z"}
{"event":"updatePeer","system":"dht",
 "peerID":"QmepsDPxWtLDuKvEoafkpJxGij4kMax11uTH7WnKqD25Dq",
 "session":"7770b5e0-25ec-47cd-aa64-f42e65a10023",
 "time":"2016-08-22T13:25:27.435843012Z"}
...
```

Returns *iterable*

version(***kwargs*)

Returns the software version of the currently connected node.

```
>>> c.version()
{'Version': '0.4.3-rc2', 'Repo': '4', 'Commit': '',
 'System': 'amd64/linux', 'Golang': 'go1.6.2'}
```

Returns **dict** (*Daemon and system version information*)

files_cp(*source*, *dest*, ***kwargs*)

Copies files within the MFS.

Due to the nature of IPFS this will not actually involve any of the file's content being copied.

```
>>> c.files_ls("/")
{'Entries': [
    {'Size': 0, 'Hash': '', 'Name': 'Software', 'Type': 0},
    {'Size': 0, 'Hash': '', 'Name': 'test', 'Type': 0}
]}
>>> c.files_cp("/test", "/bla")
```

```
''
>>> c.files_ls("/")
{'Entries': [
    {'Size': 0, 'Hash': '', 'Name': 'Software', 'Type': 0},
    {'Size': 0, 'Hash': '', 'Name': 'bla', 'Type': 0},
    {'Size': 0, 'Hash': '', 'Name': 'test', 'Type': 0}
]}
```

Parameters

- **source** (*str*) – Filepath within the MFS to copy from
- **dest** (*str*) – Destination filepath with the MFS to which the file will be copied to

files_ls(*path*, ***kwargs*)

Lists contents of a directory in the MFS.

```
>>> c.files_ls("/")
{'Entries': [
    {'Size': 0, 'Hash': '', 'Name': 'Software', 'Type': 0}
]}
```

Parameters **path** (*str*) – Filepath within the MFS

Returns **dict** (*Directory entries*)

files_mkdir(*path*, *parents=False*, ***kwargs*)

Creates a directory within the MFS.

```
>>> c.files_mkdir("/test")
b''
```

Parameters

- **path** (*str*) – Filepath within the MFS
- **parents** (*bool*) – Create parent directories as needed and do not raise an exception if the requested directory already exists

files_stat(*path*, ***kwargs*)

Returns basic `stat` information for an MFS file (including its hash).

```
>>> c.files_stat("/test")
{'Hash': 'QmUNLLsPACCz1vLxQVkXqqLX5R1X345qqfHbsf67hvA3Nn',
 'Size': 0, 'CumulativeSize': 4, 'Type': 'directory', 'Blocks': 0}
```

Parameters **path** (*str*) – Filepath within the MFS

Returns **dict** (*MFS file information*)

files_rm(*path*, *recursive=False*, ***kwargs*)

Removes a file from the MFS.

```
>>> c.files_rm("/bla/file")
b''
```

Parameters

- **path** (*str*) – Filepath within the MFS
- **recursive** (*bool*) – Recursively remove directories?

files_read(*path*, *offset=0*, *count=None*, ***kwargs*)

Reads a file stored in the MFS.

```
>>> c.files_read("/bla/file")
b'hi'
```

Parameters

- **path** (*str*) – Filepath within the MFS
- **offset** (*int*) – Byte offset at which to begin reading at
- **count** (*int*) – Maximum number of bytes to read

Returns **str** (*MFS file contents*)

files_write(*path*, *file*, *offset=0*, *create=False*, *truncate=False*, *count=None*, ***kwargs*)

Writes to a mutable file in the MFS.

```
>>> c.files_write("/test/file", io.BytesIO(b"hi"), create=True)
b''
```

Parameters

- **path** (*str*) – Filepath within the MFS
- **file** (*io.RawIOBase*) – IO stream object with data that should be written
- **offset** (*int*) – Byte offset at which to begin writing at
- **create** (*bool*) – Create the file if it does not exist
- **truncate** (*bool*) – Truncate the file to size zero before writing
- **count** (*int*) – Maximum number of bytes to read from the source `file`

files_mv(*source*, *dest*, ***kwargs*)

Moves files and directories within the MFS.

```
>>> c.files_mv("/test/file", "/bla/file")
b''
```

Parameters

- **source** (*str*) – Existing filepath within the MFS
- **dest** (*str*) – Destination to which the file will be moved in the MFS

add_bytes(**args*, ***kwargs*)

Adds a set of bytes as a file to IPFS.

```
>>> c.add_bytes(b"Mary had a little lamb")
'QmZfF6C9j4VtoCsTp4KSrhYH47QMd3DNXVZBKaxJdhaPab'
```

Also accepts and will stream generator objects.

Parameters **data** (*bytes*) – Content to be added as a file

Returns **str** (*Hash of the added IPFS object*)

add_str(**args*, ***kwargs*)
Adds a Python string as a file to IPFS.

```
>>> c.add_str(u"Mary had a little lamb")
'QmZfF6C9j4VtoCsTp4KSrhYH47QMd3DNXVZBKaxJdhaPab'
```

Also accepts and will stream generator objects.

Parameters **string** (*str*) – Content to be added as a file

Returns **str** (*Hash of the added IPFS object*)

add_json(*json_obj*, ***kwargs*)
Adds a json-serializable Python dict as a json file to IPFS.

```
>>> c.add_json({'one': 1, 'two': 2, 'three': 3})
'QmVz9g7m5u3oHiNKHj2CJX1dbG1gtismRS3g9NaPBBLbob'
```

Parameters **json_obj** (*dict*) – A json-serializable Python dictionary

Returns **str** (*Hash of the added IPFS object*)

get_json(*multihash*, ***kwargs*)
Loads a json object from IPFS.

```
>>> c.get_json('QmVz9g7m5u3oHiNKHj2CJX1dbG1gtismRS3g9NaPBBLbob')
{'one': 1, 'two': 2, 'three': 3}
```

Parameters **multihash** (*str*) – Multihash of the IPFS object to load

Returns **object** (*Deserialized IPFS JSON object value*)

add_pyobj(*py_obj*, ***kwargs*)
Adds a picklable Python object as a file to IPFS.

```
>>> c.add_pyobj([0, 1.0, 2j, '3', 4e5])
'QmWgXZSUTNNDD8LdkdJ8UXSn55KfFnNvTP1r7SyaQd74Ji'
```

Parameters **py_obj** (*object*) – A picklable Python object

Returns **str** (*Hash of the added IPFS object*)

get_pyobj(*multihash*, ***kwargs*)
Loads a pickled Python object from IPFS.

Caution: The pickle module is not intended to be secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

See the pickle module documentation for more information.

```
>>> c.get_pyobj('QmWgXZSUTNNDD8LdkdJ8UXSn55KfFnNvTP1r7SyaQd74Ji')
[0, 1.0, 2j, '3', 400000.0]
```

Parameters **multihash** (*str*) – Multihash of the IPFS object to load

Returns **object** (*Deserialized IPFS Python object*)

Internal API Reference

encoding

Defines encoding related classes.

Note: The XML and ProtoBuf encoders are currently not functional.

class ipfsapi.encoding.**Encoding**
Bases: object

Abstract base for a data parser/encoder interface.

parse_partial(*raw*)
Parses the given data and yields all complete data sets that can be built from this.

Raises *DecodingError*

Parameters **raw** (*bytes*) – Data to be parsed

Returns *generator*

parse_finalize()
Finalizes parsing based on remaining buffered data and yields the remaining data sets.

Raises *DecodingError*

Returns *generator*

parse(*raw*)
Returns a Python object decoded from the bytes of this encoding.

Raises *DecodingError*

Parameters **raw** (*bytes*) – Data to be parsed

Returns *object*

encode(*obj*)
Serialize a raw object into corresponding encoding.

Raises *EncodingError*

Parameters **obj** (*object*) – Object to be encoded

class ipfsapi.encoding.**Dummy**
Bases: *ipfsapi.encoding.Encoding*

Dummy parser/encoder that does nothing.

parse_partial(*raw*)
Yields the data passed into this method.

Parameters **raw** (*bytes*) – Any kind of data

Returns *generator*

encode(*obj*)
Returns the bytes representation of the data passed into this function.

Parameters **obj** (*object*) – Any Python object

Returns *bytes*

class ipfsapi.encoding.**Json**
Bases: *ipfsapi.encoding.Encoding*

JSON parser/encoder that handles concatenated JSON.

parse_partial(*data*)
Incrementally decodes JSON data sets into Python objects.

Raises *DecodingError*

Returns *generator*

parse_finalize()
Raises errors for incomplete buffered data that could not be parsed because the end of the input data has been reached.

Raises *DecodingError*

Returns **tuple** (*Always empty*)

encode(*obj*)
Returns `obj` serialized as JSON formatted bytes.

Raises *EncodingError*

Parameters **obj** (*str | list | dict | int*) – JSON serializable Python object

Returns *bytes*

class ipfsapi.encoding.**Pickle**
Bases: *ipfsapi.encoding.Encoding*

Python object parser/encoder using *pickle*.

parse_partial(*raw*)
Buffers the given data so that the it can be passed to *pickle* in one go.

This does not actually process the data in smaller chunks, but merely buffers it until *parse_finalize* is called! This is mostly because the standard-library module expects the entire data to be available up front, which is currently always the case for our code anyways.

Parameters **raw** (*bytes*) – Data to be buffered

Returns **tuple** (*An empty tuple*)

parse_finalize()
Parses the buffered data and yields the result.

Raises *DecodingError*

Returns *generator*

parse(*raw*)
Returns a Python object decoded from a pickle byte stream.

```
>>> p = Pickle()
>>> p.parse(b'(lp0\nI1\naI2\naI3\naI01\naF4.5\naNaF6000.0\na.')
[1, 2, 3, True, 4.5, None, 6000.0]
```

Raises *DecodingError*

Parameters **raw** (*bytes*) – Pickle data bytes

Returns *object*

encode(*obj*)
Returns `obj` serialized as a pickle binary string.

Raises *EncodingError*

Parameters **obj** (*object*) – Serializable Python object

Returns *bytes*

class ipfsapi.encoding.**Protobuf**
Bases: *ipfsapi.encoding.Encoding*

Protobuf parser/encoder that handles protobuf.

class ipfsapi.encoding.**Xml**
Bases: *ipfsapi.encoding.Encoding*

XML parser/encoder that handles XML.

ipfsapi.encoding.**get_encoding**(*name*)
Returns an Encoder object for the named encoding

Raises *EncoderMissingError*

Parameters **name** (*str*) – Encoding name. Supported options:

- `"none"`
- `"json"`
- `"pickle"`
- `"protobuf"`
- `"xml"`

http

HTTP client for api requests.

This is pluggable into the IPFS Api client and will hopefully be supplemented by an asynchronous version.

ipfsapi.http.**pass_defaults**(*func*)
Decorator that returns a function named wrapper.

When invoked, wrapper invokes func with default kwargs appended.

Parameters **func** (*callable*) – The function to append the default kwargs to

class ipfsapi.http.**HTTPClient**(*host*, *port*, *base*, ***defaults*)
Bases: `object`

An HTTP client for interacting with the IPFS daemon.

Parameters

- **host** (*str*) – The host the IPFS daemon is running on
- **port** (*int*) – The port the IPFS daemon is running at
- **base** (*str*) – The path prefix for API calls

- **defaults** (*dict*) – The default parameters to be passed to *request()*

request(**args*, ***kwargs*)

Makes an HTTP request to the IPFS daemon.

This function returns the contents of the HTTP response from the IPFS daemon.

Raises

- *ErrorResponse*
- *ConnectionError*
- *ProtocolError*
- *StatusError*
- *TimeoutError*

Parameters

- **path** (*str*) – The REST command path to send
- **args** (*list*) – Positional parameters to be sent along with the HTTP request
- **files** (io.RawIOBase | str | list) – The file object(s) or path(s) to stream to the daemon
- **opts** (*dict*) – Query string paramters to be sent along with the HTTP request
- **decoder** (*str*) – The encoder to use to parse the HTTP response
- **kwargs** (*dict*) – Additional arguments to pass to `requests`

download(**args*, ***kwargs*)

Makes a request to the IPFS daemon to download a file.

Downloads a file or files from IPFS into the current working directory, or the directory given by `filepath`.

Raises

- *ErrorResponse*
- *ConnectionError*
- *ProtocolError*
- *StatusError*
- *TimeoutError*

Parameters

- **path** (*str*) – The REST command path to send
- **filepath** (*str*) – The local path where IPFS will store downloaded files

  Defaults to the current working directory.
- **args** (*list*) – Positional parameters to be sent along with the HTTP request
- **opts** (*dict*) – Query string paramters to be sent along with the HTTP request
- **compress** (*bool*) – Whether the downloaded file should be GZip compressed by the daemon before being sent to the client
- **kwargs** (*dict*) – Additional arguments to pass to `requests`

session (**args*, ***kwds*)
A context manager for this client's session.

This function closes the current session when this client goes out of scope.

multipart

HTTP *multipart/**-encoded file streaming.

ipfsapi.multipart.**content_disposition** (*fn*, *disptype='file'*)
Returns a dict containing the MIME content-disposition header for a file.

```
>>> content_disposition('example.txt')
{'Content-Disposition': 'file; filename="example.txt"'}

>>> content_disposition('example.txt', 'attachment')
{'Content-Disposition': 'attachment; filename="example.txt"'}
```

Parameters

- **fn** (*str*) – Filename to retrieve the MIME content-disposition for
- **disptype** (*str*) – Rhe disposition type to use for the file

ipfsapi.multipart.**content_type** (*fn*)
Returns a dict with the content-type header for a file.

Guesses the mimetype for a filename and returns a dict containing the content-type header.

```
>>> content_type('example.txt')
{'Content-Type': 'text/plain'}

>>> content_type('example.jpeg')
{'Content-Type': 'image/jpeg'}

>>> content_type('example')
{'Content-Type': 'application/octet-stream'}
```

Parameters **fn** (*str*) – Filename to guess the content-type for

ipfsapi.multipart.**multipart_content_type** (*boundary*, *subtype='mixed'*)
Creates a MIME multipart header with the given configuration.

Returns a dict containing a MIME multipart header with the given boundary.

```
>>> multipart_content_type('8K5rNKlLQVyreRNncxOTeg')
{'Content-Type': 'multipart/mixed; boundary="8K5rNKlLQVyreRNncxOTeg"'}

>>> multipart_content_type('8K5rNKlLQVyreRNncxOTeg', 'alt')
{'Content-Type': 'multipart/alt; boundary="8K5rNKlLQVyreRNncxOTeg"'}
```

Parameters

- **boundry** (*str*) – The content delimiter to put into the header
- **subtype** (*str*) – The subtype in *multipart/**-domain to put into the header

class ipfsapi.multipart.**BodyGenerator**(*name*, *disptype='file'*, *subtype='mixed'*, *boundary=None*)

Bases: object

Generators for creating the body of a *multipart/** HTTP request.

Parameters

- **name** (*str*) – The filename of the file(s)/content being encoded
- **disptype** (*str*) – The Content-Disposition of the content
- **subtype** (*str*) – The *multipart/**-subtype of the content
- **boundary** (*str*) – An identifier used as a delimiter for the content's body

write_headers()
Yields the HTTP header text for the content.

open(***kwargs*)
Yields the body section for the content.

file_open(*fn*)
Yields the opening text of a file section in multipart HTTP.

Parameters **fn** (*str*) – Filename for the file being opened and added to the HTTP body

file_close()
Yields the end text of a file section in HTTP multipart encoding.

close()
Yields the ends of the content area in a HTTP multipart body.

class ipfsapi.multipart.**BufferedGenerator**(*name*, *chunk_size=4096*)

Bases: object

Generator that encodes multipart/form-data.

An abstract buffered generator class which encodes *multipart/form-data*.

Parameters

- **name** (*str*) – The name of the file to encode
- **chunk_size** (*int*) – The maximum size that any single file chunk may have in bytes

file_chunks(*fp*)
Yields chunks of a file.

Parameters **fp** (*io.RawIOBase*) – The file to break into chunks (must be an open file or have the readinto method)

gen_chunks(*gen*)
Generates byte chunks of a given size.

Takes a bytes generator and yields chunks of a maximum of chunk_size bytes.

Parameters **gen** (*generator*) – The bytes generator that produces the bytes

body(**args*, ***kwargs*)
Returns the body of the buffered file.

Note: This function is not actually implemented.

---

close()
Yields the closing text of a multipart envelope.

class ipfsapi.multipart.**FileStream**(*files*, *chunk_size=4096*)
Bases: *ipfsapi.multipart.BufferedGenerator*

Generator that encodes multiples files into HTTP multipart.

A buffered generator that encodes an array of files as *multipart/form-data*. This is a concrete implementation of *BufferedGenerator*.

Parameters

- **name** (*str*) – The filename of the file to encode
- **chunk_size** (*int*) – The maximum size that any single file chunk may have in bytes

body()
Yields the body of the buffered file.

ipfsapi.multipart.**glob_compile**(*pat*)
Translate a shell glob PATTERN to a regular expression.

This is almost entirely based on *fnmatch.translate* source-code from the python 3.5 standard-library.

class ipfsapi.multipart.**DirectoryStream**(*directory*, *recursive=False*, *patterns='**'*, *chunk_size=4096*)
Bases: *ipfsapi.multipart.BufferedGenerator*

Generator that encodes a directory into HTTP multipart.

A buffered generator that encodes an array of files as *multipart/form-data*. This is a concrete implementation of *BufferedGenerator*.

Parameters

- **directory** (*str*) – The filepath of the directory to encode
- **patterns** (*str | list*) – A single glob pattern or a list of several glob patterns and compiled regular expressions used to determine which filepaths to match
- **chunk_size** (*int*) – The maximum size that any single file chunk may have in bytes

body()
Returns the HTTP headers for this directory upload request.

headers()
Returns the HTTP body for this directory upload request.

class ipfsapi.multipart.**BytesStream**(*data*, *chunk_size=4096*)
Bases: *ipfsapi.multipart.BufferedGenerator*

A buffered generator that encodes bytes as *multipart/form-data*.

Parameters

- **data** (*bytes*) – The binary data to stream to the daemon
- **chunk_size** (*int*) – The maximum size of a single data chunk

body()
Yields the encoded body.

ipfsapi.multipart.**stream_files**(*files*, *chunk_size=4096*)
Gets a buffered generator for streaming files.

Returns a buffered generator which encodes a file or list of files as `multipart/form-data` with the corresponding headers.

Parameters

- **files** (*str*) – The file(s) to stream
- **chunk_size** (*int*) – Maximum size of each stream chunk

ipfsapi.multipart.**stream_directory**(*directory*, *recursive=False*, *patterns='**'*, *chunk_size=4096*)

Gets a buffered generator for streaming directories.

Returns a buffered generator which encodes a directory as `multipart/form-data` with the corresponding headers.

Parameters

- **directory** (*str*) – The filepath of the directory to stream
- **recursive** (*bool*) – Stream all content within the directory recursively?
- **patterns** (*str | list*) – Single *glob* pattern or list of *glob* patterns and compiled regular expressions to match the names of the filepaths to keep
- **chunk_size** (*int*) – Maximum size of each stream chunk

ipfsapi.multipart.**stream_filesystem_node**(*path*, *recursive=False*, *patterns='**'*, *chunk_size=4096*)

Gets a buffered generator for streaming either files or directories.

Returns a buffered generator which encodes the file or directory at the given path as `multipart/form-data` with the corresponding headers.

Parameters

- **path** (*str*) – The filepath of the directory or file to stream
- **recursive** (*bool*) – Stream all content within the directory recursively?
- **patterns** (*str | list*) – Single *glob* pattern or list of *glob* patterns and compiled regular expressions to match the names of the filepaths to keep
- **chunk_size** (*int*) – Maximum size of each stream chunk

ipfsapi.multipart.**stream_bytes**(*data*, *chunk_size=4096*)

Gets a buffered generator for streaming binary data.

Returns a buffered generator which encodes binary data as `multipart/form-data` with the corresponding headers.

Parameters

- **data** (*bytes*) – The data bytes to stream
- **chunk_size** (*int*) – The maximum size of each stream chunk

Returns *(generator, dict)*

ipfsapi.multipart.**stream_text**(*text*, *chunk_size=4096*)

Gets a buffered generator for streaming text.

Returns a buffered generator which encodes a string as `multipart/form-data` with the corresponding headers.

Parameters

- **text** (*str*) – The data bytes to stream

- **chunk_size** (*int*) – The maximum size of each stream chunk

Returns *(generator, dict)*

utils

A module to handle generic operations.

ipfsapi.utils.**guess_mimetype**(*filename*)
Guesses the mimetype of a file based on the given `filename`.

```
>>> guess_mimetype('example.txt')
'text/plain'
>>> guess_mimetype('/foo/bar/example')
'application/octet-stream'
```

Parameters **filename** (*str*) – The file name or path for which the mimetype is to be guessed

ipfsapi.utils.**ls_dir**(*dirname*)
Returns files and subdirectories within a given directory.

Returns a pair of lists, containing the names of directories and files in `dirname`.

Raises OSError : Accessing the given directory path failed

Parameters **dirname** (*str*) – The path of the directory to be listed

ipfsapi.utils.**clean_file**(*file*)
Returns a tuple containing a `file`-like object and a close indicator.

This ensures the given file is opened and keeps track of files that should be closed after use (files that were not open prior to this function call).

Raises OSError : Accessing the given file path failed

Parameters **file** (*str | io.IOBase*) – A filepath or `file`-like object that may or may not need to be opened

ipfsapi.utils.**clean_files**(*files*)
Generates tuples with a `file`-like object and a close indicator.

This is a generator of tuples, where the first element is the file object and the second element is a boolean which is True if this module opened the file (and thus should close it).

Raises OSError : Accessing the given file path failed

Parameters **files** (*list | io.IOBase | str*) – Collection or single instance of a filepath and file-like object

ipfsapi.utils.**file_size**(*f*)
Returns the size of a file in bytes.

Raises OSError : Accessing the given file path failed

Parameters **f** (*io.IOBase | str*) – The file path or object for which the size should be determined

class ipfsapi.utils.**return_field**(*field*)
Bases: `object`

Decorator that returns the given field of a json response.

Parameters **field** (*object*) – The response field to be returned for all invocations

__call__(*cmd*)
Wraps a command so that only a specified field is returned.

Parameters **cmd** (*callable*) – A command that is intended to be wrapped

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

ipfsapi.encoding, 24
ipfsapi.exceptions, 1
ipfsapi.http, 26
ipfsapi.multipart, 28
ipfsapi.utils, 32

Index

Symbols

__call__() (ipfsapi.utils.return_field method), 33

A

add() (ipfsapi.Client method), 3
add_bytes() (ipfsapi.Client method), 22
add_json() (ipfsapi.Client method), 23
add_pyobj() (ipfsapi.Client method), 23
add_str() (ipfsapi.Client method), 23
assert_version() (in module ipfsapi), 2

B

bitswap_stat() (ipfsapi.Client method), 5
bitswap_unwant() (ipfsapi.Client method), 5
bitswap_wantlist() (ipfsapi.Client method), 5
block_get() (ipfsapi.Client method), 4
block_put() (ipfsapi.Client method), 5
block_stat() (ipfsapi.Client method), 4
body() (ipfsapi.multipart.BufferedGenerator method), 29
body() (ipfsapi.multipart.BytesStream method), 30
body() (ipfsapi.multipart.DirectoryStream method), 30
body() (ipfsapi.multipart.FileStream method), 30
BodyGenerator (class in ipfsapi.multipart), 28
bootstrap() (ipfsapi.Client method), 14
bootstrap_add() (ipfsapi.Client method), 14
bootstrap_list() (ipfsapi.Client method), 14
bootstrap_rm() (ipfsapi.Client method), 14
BufferedGenerator (class in ipfsapi.multipart), 29
BytesStream (class in ipfsapi.multipart), 30

C

cat() (ipfsapi.Client method), 3
clean_file() (in module ipfsapi.utils), 32
clean_files() (in module ipfsapi.utils), 32
Client (class in ipfsapi), 3
close() (ipfsapi.multipart.BodyGenerator method), 29
close() (ipfsapi.multipart.BufferedGenerator method), 29
CommunicationError, 2
config() (ipfsapi.Client method), 18
config_replace() (ipfsapi.Client method), 19
config_show() (ipfsapi.Client method), 18
connect() (in module ipfsapi), 2
ConnectionError, 2
content_disposition() (in module ipfsapi.multipart), 28
content_type() (in module ipfsapi.multipart), 28

D

DecodingError, 2
dht_findpeer() (ipfsapi.Client method), 17
dht_findprovs() (ipfsapi.Client method), 16
dht_get() (ipfsapi.Client method), 17
dht_put() (ipfsapi.Client method), 17
dht_query() (ipfsapi.Client method), 16
DirectoryStream (class in ipfsapi.multipart), 30
dns() (ipfsapi.Client method), 11
download() (ipfsapi.http.HTTPClient method), 27
Dummy (class in ipfsapi.encoding), 24

E

encode() (ipfsapi.encoding.Dummy method), 24
encode() (ipfsapi.encoding.Encoding method), 24
encode() (ipfsapi.encoding.Json method), 25
encode() (ipfsapi.encoding.Pickle method), 26
EncoderError, 1
EncoderMissingError, 1
Encoding (class in ipfsapi.encoding), 24
EncodingError, 1
Error, 1
ErrorResponse, 2

F

file_chunks() (ipfsapi.multipart.BufferedGenerator method), 29
file_close() (ipfsapi.multipart.BodyGenerator method), 29
file_ls() (ipfsapi.Client method), 9
file_open() (ipfsapi.multipart.BodyGenerator method), 29
file_size() (in module ipfsapi.utils), 32

files_cp() (ipfsapi.Client method), 20
files_ls() (ipfsapi.Client method), 21
files_mkdir() (ipfsapi.Client method), 21
files_mv() (ipfsapi.Client method), 22
files_read() (ipfsapi.Client method), 22
files_rm() (ipfsapi.Client method), 21
files_stat() (ipfsapi.Client method), 21
files_write() (ipfsapi.Client method), 22
FileStream (class in ipfsapi.multipart), 30

G

gen_chunks() (ipfsapi.multipart.BufferedGenerator method), 29
get() (ipfsapi.Client method), 3
get_encoding() (in module ipfsapi.encoding), 26
get_json() (ipfsapi.Client method), 23
get_pyobj() (ipfsapi.Client method), 23
glob_compile() (in module ipfsapi.multipart), 30
guess_mimetype() (in module ipfsapi.utils), 32

H

headers() (ipfsapi.multipart.DirectoryStream method), 30
HTTPClient (class in ipfsapi.http), 26

I

id() (ipfsapi.Client method), 13
ipfsapi.encoding (module), 24
ipfsapi.exceptions (module), 1
ipfsapi.http (module), 26
ipfsapi.multipart (module), 28
ipfsapi.utils (module), 32

J

Json (class in ipfsapi.encoding), 25

K

key_gen() (ipfsapi.Client method), 10
key_list() (ipfsapi.Client method), 10

L

log_level() (ipfsapi.Client method), 19
log_ls() (ipfsapi.Client method), 19
log_tail() (ipfsapi.Client method), 20
ls() (ipfsapi.Client method), 3
ls_dir() (in module ipfsapi.utils), 32

M

multipart_content_type() (in module ipfsapi.multipart), 28

N

name_publish() (ipfsapi.Client method), 10
name_resolve() (ipfsapi.Client method), 11

O

object_data() (ipfsapi.Client method), 5
object_get() (ipfsapi.Client method), 6
object_links() (ipfsapi.Client method), 6
object_new() (ipfsapi.Client method), 6
object_patch_add_link() (ipfsapi.Client method), 8
object_patch_append_data() (ipfsapi.Client method), 7
object_patch_rm_link() (ipfsapi.Client method), 8
object_patch_set_data() (ipfsapi.Client method), 8
object_put() (ipfsapi.Client method), 7
object_stat() (ipfsapi.Client method), 7
open() (ipfsapi.multipart.BodyGenerator method), 29

P

parse() (ipfsapi.encoding.Encoding method), 24
parse() (ipfsapi.encoding.Pickle method), 25
parse_finalize() (ipfsapi.encoding.Encoding method), 24
parse_finalize() (ipfsapi.encoding.Json method), 25
parse_finalize() (ipfsapi.encoding.Pickle method), 25
parse_partial() (ipfsapi.encoding.Dummy method), 24
parse_partial() (ipfsapi.encoding.Encoding method), 24
parse_partial() (ipfsapi.encoding.Json method), 25
parse_partial() (ipfsapi.encoding.Pickle method), 25
pass_defaults() (in module ipfsapi.http), 26
Pickle (class in ipfsapi.encoding), 25
pin_add() (ipfsapi.Client method), 12
pin_ls() (ipfsapi.Client method), 12
pin_rm() (ipfsapi.Client method), 12
ping() (ipfsapi.Client method), 18
Protobuf (class in ipfsapi.encoding), 26
ProtocolError, 2

R

refs() (ipfsapi.Client method), 4
refs_local() (ipfsapi.Client method), 4
repo_gc() (ipfsapi.Client method), 13
repo_stat() (ipfsapi.Client method), 13
request() (ipfsapi.http.HTTPClient method), 27
resolve() (ipfsapi.Client method), 9
return_field (class in ipfsapi.utils), 32

S

session() (ipfsapi.http.HTTPClient method), 27
StatusError, 2
stream_bytes() (in module ipfsapi.multipart), 31
stream_directory() (in module ipfsapi.multipart), 31
stream_files() (in module ipfsapi.multipart), 30
stream_filesystem_node() (in module ipfsapi.multipart), 31
stream_text() (in module ipfsapi.multipart), 31
swarm_addrs() (ipfsapi.Client method), 14
swarm_connect() (ipfsapi.Client method), 15
swarm_disconnect() (ipfsapi.Client method), 15

swarm_filters_add() (ipfsapi.Client method), 15
swarm_filters_rm() (ipfsapi.Client method), 16
swarm_peers() (ipfsapi.Client method), 14

T

TimeoutError, 2

V

version() (ipfsapi.Client method), 20
VersionMismatch, 1

W

write_headers() (ipfsapi.multipart.BodyGenerator method), 29

X

Xml (class in ipfsapi.encoding), 26