bushel Documentation

Release 0.0.0a

Tor Project

May 20, 2019

Contents:

A bushel of onions is 57lbs.

bushel contains a number of tools for interacting with Tor networks and Tor-related data. Many design decisions have been taken to specifically benefit the use-cases of Tor Metrics. If you are looking for a general-purpose library for working with Tor, you may instead want to look at stem.

CHAPTER 1

Directory Archive

Persistent filesystem-backed archive for Tor directory protocol descriptors. This is intended to be used as part of an `asyncio` application. File I/O operations are provided by coroutines and coroutine methods, with the actual I/O performed in an executor.

class bushel.archive.**CollectorOutBridgeDescsMarker**

Enumeration of marker names under the "bridge-descriptors" directory as specified in §5.2 of [collector-protocol].

Name	Description
EXTRA_INFO	Bridge extra-info descriptors (§5.2.1)
SERVER_DESCRIPTOR	Bridge server descriptors (§5.2.1)
STATUS	Bridge statuses (§5.2.2)

class bushel.archive.**CollectorOutRelayDescsMarker**

Enumeration of marker names under the "relay-descriptors" directory as specified in §5.3 of [collector-protocol].

Name	Description
CONSENSUS	Network status consensuses (§5.3.2)
EXTRA_INFO	Relay extra-info descriptors (§5.3.2)
SERVER_DESCRIPTOR	Relay server descriptors (§5.3.2)
VOTE	Network status votes (§5.3.2)

class bushel.archive.**CollectorOutSubdirectory**

Enumeration of subdirectory names under the "out" directory as specified in §5.0 of [collector-protocol].

Name	Description
BRIDGE_DESCRIPTORS	Bridge descriptors (§5.2)
EXIT_LISTS	Exit lists (§5.1)
RELAY_DESCRIPTORS	Relay descriptors (§5.3)
TORPERF	Torperf and Onionperf (§5.1)
WEBSTATS	Web server access logs (§5.4)

class bushel.archive.**DirectoryArchive**(*archive_path*, *max_file_concurrency=100*)
Persistent filesystem-backed archive for Tor directory protocol descriptors.

This implements the CollecTor File Structure Protocol as detailed in [collector-protocol].

Parameters **archive_path** (*str*) – Either an absolute or relative path to the location of the directory to use for the archive. This location must exist, but may be an empty directory.

bridge_extra_info_descriptor_path(*published*, *digest*)
Generates a path, including the archive path, for a bridge extra-info descriptor with a given published time and digest. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> archive.bridge_extra_info_descriptor_path(published, digest)  # doctest:␣
↪+ELLIPSIS
'/srv/archive/bridge-descriptors/extra-info/2018/11/a/9/a94a...389'
```

These paths are defined in §5.2.1 of [collector-protocol].

Parameters

- **published** (*datetime*) – The published time of the descriptor.
- **digest** (*str*) – The hex-encoded SHA-1 digest of the descriptor.

Returns Archive path as a str.

bridge_server_descriptor_path(*published*, *digest*)
Generates a path, including the archive path, for a bridge server descriptor with a given published time and digest. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> published = datetime.datetime(2018, 11, 19, 15, 1, 2)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> archive.bridge_server_descriptor_path(published, digest)  # doctest:␣
↪+ELLIPSIS
'/srv/archive/bridge-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

These paths are defined in §5.2.1 of [collector-protocol].

Parameters

- **published** (*datetime*) – The published time of the descriptor.
- **digest** (*str*) – The hex-encoded SHA-1 digest of the descriptor.

Returns Archive path as a str.

bridge_status_path(*valid_after*, *fingerprint*)
Generates a path, including the archive path, for a bridge status valid-after time and generated by the authority with the given fingerprint. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> fingerprint = "BA44A889E64B93FAA2B114E02C2A279A8555C533"  # Serge
>>> archive.bridge_status_path(valid_after, fingerprint)  # doctest: +ELLIPSIS
'/srv/archive/bridge-descriptors/statuses/2018/11/19/20181119-150000-BA...33'
```

These paths are defined in §5.2.2 of [collector-protocol].

Parameters

- **valid_after** (*datetime*) – The valid-after time for the status.
- **fingerprint** (*str*) – The fingerprint of the bridge authority.

Returns Path as a `str`.

path_for(*descriptor*, *create_dir=False*)

The filesystem path that a descriptor will be archived at. These paths are defined in [collector-protocol].

It is also possible to set *descriptor* with a `str` in which case it will be treated as a relative path from the root of the archive. For example:

```
>>> DirectoryArchive("/srv/archive").path_for("path/to/descriptor")
'/srv/archive/path/to/descriptor'
```

Parameters **create_dir** (*bool*) – Create the directory ready to archive a descriptor.

Returns Archive path for the descriptor as a `str`.

relay_consensus(*flavor='ns'*, *valid_after=None*)

Retrieves a consensus from the archive.

Parameters **valid_after** (*datetime*) – If set, will retrieve a consensus with the given valid_after time, otherwise a vote that became valid at the top of the current hour will be retrieved.

Returns A `NetworkStatusDocumentV3` if found, otherwise *None*.

relay_consensus_path(*valid_after*)

Generates a path, including the archive path, for a network-status consensus with a given valid-after time. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> archive.relay_consensus_path(valid_after)
'/srv/archive/relay-descriptors/consensus/2018/11/19/2018-11-19-15-00-00-
↪consensus'
```

These paths are defined in §5.3.2 of [collector-protocol].

Parameters

- **valid_after** (*datetime*) – The valid-after time for the status.
- **fingerprint** (*str*) – The fingerprint of the bridge authority.

Returns Path as a `str`.

relay_extra_info_descriptor(*digest*, *published_hint*)

Retrieves a relay's extra-info descriptor from the archive.

Parameters

- **digest** (*str*) – A hex-encoded digest of the descriptor.
- **published_hint** (*datetime*) – Provides a hint on the published time to allow the descriptor to be found in the archive. If the descriptor was not published in the same month as this, it will not be found.

Returns A `RelayExtraInfoDescriptor` if found, otherwise *None*.

relay_extra_info_descriptor_path(*published*, *digest*)
Generates a path, including the archive path, for a relay extra-info descriptor with a given published time and digest. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> archive.relay_extra_info_descriptor_path(published, digest)  # doctest:␣
↪+ELLIPSIS
'/srv/archive/relay-descriptors/extra-info/2018/11/a/9/a94a...389'
```

These paths are defined in §5.3.2 of [collector-protocol].

Parameters

- **published** (*datetime*) – The published time of the descriptor.
- **digest** (*str*) – The hex-encoded SHA-1 digest of the descriptor.

Returns Path as a `str`.

relay_extra_info_descriptors(*digests*, *published_hint*)
Retrieves multiple extra-info descriptors published around the same time (e.g. all referenced by server-descriptors in the same consensus).

Parameters

- **digest** (*list(str)*) – Hex-encoded digests for the descriptors.
- **published_hint** (*datetime*) – Provides a hint on the published time to allow the descriptor to be found in the archive. If the descriptor was not published in the same month as this, it will not be found.

Returns A `list` of `stem.descriptor.extrainfo_descriptor.RelayExtraInfoDescriptor`.

relay_microdescriptor(*digest*, *valid_after_hint*)
Retrieves a relay's microdescriptor from the archive.

Parameters

- **digest** (*str*) – A hex-encoded digest of the descriptor.
- **valid_after_hint** (*datetime*) – Provides a hint on the valid_after time to allow the descriptor to be found in the archive. If the descriptor did not become valid in the same month as this, it will not be found.

Returns A `stem.descriptor.microdescriptor.Microdescriptor` if found, otherwise *None*.

relay_microdescriptors(*digests*, *valid_after_hint*)
Retrieves multiple microdescriptors around the same valid_after time (e.g. all referenced by the same microdescriptor consensus).

Parameters

- **digest** (*list(str)*) – Hex-encoded digests for the descriptors.
- **valid_after_hint** (*datetime*) – Provides a hint on the valid_after time to allow the descriptor to be found in the archive. If the descriptor did not become valid in the same month as this, it will not be found.

Returns A `list` of `stem.descriptor.microdescriptor.Microdescriptor`.

relay_server_descriptor(*digest*, *published_hint*)
Retrieves a relay's server descriptor from the archive.

Parameters

- **digest** (*str*) – A hex-encoded digest of the descriptor.
- **published_hint** (*datetime*) – Provides a hint on the published time to allow the descriptor to be found in the archive. If the descriptor was not published in the same month as this, it will not be found.

Returns A stem.descriptor.server_descriptor.RelayDescriptor if found, otherwise *None*.

relay_server_descriptor_path(*published*, *digest*)
Generates a path, including the archive path, for a relay server descriptor with a given published time and digest. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> published = datetime.datetime(2018, 11, 19, 15, 1, 2)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> archive.relay_server_descriptor_path(published, digest)  # doctest:␣
↪+ELLIPSIS
'/srv/archive/relay-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

These paths are defined in §5.3.2 of [collector-protocol].

Parameters

- **published** (*datetime*) – The published time of the descriptor.
- **digest** (*str*) – The hex-encoded SHA-1 digest of the descriptor.

Returns Path as a str.

relay_server_descriptors(*digests*, *published_hint*)
Retrieves multiple server descriptors published around the same time (e.g. all referenced by the same consensus).

Parameters

- **digest** (*list (str)*) – Hex-encoded digests for the descriptors.
- **published_hint** (*datetime*) – Provides a hint on the published time to allow the descriptor to be found in the archive. If the descriptor was not published in the same month as this, it will not be found.

Returns A list of stem.descriptor.server_descriptor.RelayDescriptor.

relay_vote(*v3ident*, *digest='*'*, *valid_after=None*)
Retrieves a vote from the archive.

Parameters

- **v3ident** (*str*) – The v3ident of the authority that created the vote.
- **digest** (*str*) – A hex-encoded digest of the vote. This will automatically be fixed to upper-case.
- **valid_after** (*datetime*) – If set, will retrieve a consensus with the given valid_after time, otherwise a vote that became valid at the top of the current hour will be retrieved.

Returns A NetworkStatusDocumentV3 if found, otherwise *None*.

relay_vote_path(*valid_after*, *v3ident*, *digest*)

Generates a path, including the archive path, for a network-status vote with a given valid-after time, generated by the authority with the given v3ident, and with the given digest. For example:

```
>>> archive = DirectoryArchive("/srv/archive")
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> v3ident = "D586D18309DED4CD6D57C18FDB97EFA96D330566"  # moria1
>>> digest = "663B503182575D242B9D8A67334365FF8ECB53BB"
>>> archive.relay_vote_path(valid_after, v3ident, digest)  # doctest:␣
↪+ELLIPSIS
'/srv/archive/relay-descriptors/vote/2018/11/19/2018-11-19-15-00-00-vote-D...-
↪...B'
```

These paths are defined in §5.3.2 of [collector-protocol].

Parameters

- **valid_after** (*datetime*) – The valid-after time.
- **v3ident** (*str*) – The v3ident of the directory authority.
- **digest** (*str*) – The digest of the vote.

Returns Path as a `str`.

bushel.archive.**aglob**(*pathname*, *, *recursive=False*)

`asyncio` wrapper for `glob.glob()`.

bushel.archive.**collector_422_filename**(*valid_after*, *fingerprint*)

Create a filename for a bridge status according to §4.2.2 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> fingerprint = "BA44A889E64B93FAA2B114E02C2A279A8555C533" # Serge
>>> collector_422_filename(valid_after, fingerprint)
'20181119-150000-BA44A889E64B93FAA2B114E02C2A279A8555C533'
```

Parameters

- **valid_after** (*datetime*) – The valid-after time.
- **fingerprint** (*str*) – The fingerprint of the bridge authority.

Returns Filename as a `str`.

bushel.archive.**collector_431_filename**(*valid_after*)

Create a filename for a network status consensus according to §4.3.1 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_431_filename(valid_after)
'2018-11-19-15-00-00-consensus'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Filename as a `str`.

bushel.archive.**collector_433_filename**(*valid_after*, *v3ident*, *digest*)

Create a filename for a network status vote according to §4.3.3 of the [collector-protocol].

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> v3ident = "D586D18309DED4CD6D57C18FDB97EFA96D330566"  # moria1
>>> digest = "663B503182575D242B9D8A67334365FF8ECB53BB"
>>> collector_433_filename(valid_after, v3ident, digest)  # doctest: +ELLIPSIS
'2018-11-19-15-00-00-vote-D586D18309DED4CD6D57C18FDB97EFA96D330566-663B...3BB'
```

Paths in the Collector File Structure Protocol using this filename expect *upper-case* hex-encoded SHA-1 digests.

```
>>> v3ident = "d586d18309ded4cd6d57c18fdb97efa96d330566"  # Lower case gets␣
↪corrected
>>> digest = "663b503182575d242b9d8a67334365ff8ecb53bb"  # Lower case gets␣
↪corrected
>>> collector_433_filename(valid_after, v3ident, digest)  # doctest: +ELLIPSIS
'2018-11-19-15-00-00-vote-D586D18309DED4CD6D57C18FDB97EFA96D330566-663B...3BB'
```

Parameters

- **valid_after** (*datetime*) – The valid-after time.
- **v3ident** (*str*) – The v3ident of the directory authority.
- **digest** (*str*) – The digest of the vote.

Returns Filename as a `str`.

`bushel.archive.`**collector_434_filename**(*valid_after*)

Create a filename for a microdesc-flavoured network status consensus according to §4.3.4 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_434_filename(valid_after)
'2018-11-19-15-00-00-consensus-microdesc'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Filename as a `str`.

`bushel.archive.`**collector_521_path**(*subdirectory*, *marker*, *published*, *digest*)

Create a path according to §5.2.1 of the [collector-protocol]. This is used for server-descriptors and extra-info descriptors for both relays and bridges. For example:

```
>>> subdirectory = CollectorOutSubdirectory.RELAY_DESCRIPTORS
>>> marker = CollectorOutRelayDescsMarker.SERVER_DESCRIPTOR
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> collector_521_path(subdirectory, marker, published, digest)  # doctest:␣
↪+ELLIPSIS
'relay-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

Paths in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-1 digests.

```
>>> digest = "A94A07B201598D847105AE5FCD5BC3AB10124389" # Upper case gets␣
↪corrected
>>> collector_521_path(subdirectory, marker, published, digest)  # doctest:␣
↪+ELLIPSIS
'relay-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

Parameters

- **subdirectory** (*str*) – The subdirectory under the "out" directory to use. Standard values can be found in *CollectorOutSubdirectory*.
- **marker** (*str*) – The marker under the subdirectory to use. Standard values can be found in *CollectorOutRelayDescsMarker* and *CollectorOutBridgeDescsMarker*.
- **published** (*datetime*) – The published time.
- **digest** (*str*) – The hex-encoded SHA-1 digest for the descriptor. The case will automatically be fixed to lower-case.

Returns Path for the descriptor as a `str`.

bushel.archive.**collector_521_substructure**(*published*, *digest*)

Create a path substructure according to §5.2.1 of the [collector-protocol]. This is used for server-descriptors and extra-info descriptors for both relays and bridges. For example:

```
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> collector_521_substructure(published, digest)
'2018/11/a/9'
```

Paths in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-1 digests.

```
>>> digest = "A94A07B201598D847105AE5FCD5BC3AB10124389" # Upper case gets␣
↪corrected
>>> collector_521_substructure(published, digest)
'2018/11/a/9'
```

Parameters

- **published** (*datetime*) – The published time.
- **digest** (*str*) – The hex-encoded SHA-1 digest for the descriptor. The case will automatically be fixed to lower-case.

Returns Path substructure as a `str`.

bushel.archive.**collector_522_path**(*subdirectory*, *marker*, *valid_after*, *filename*)

Create a path according to §5.2.2 of the [collector-protocol]. This is used for bridge statuses, and network-status consensuses (both ns- and microdesc- flavors) and votes. For a bridge status for example:

```
>>> subdirectory = CollectorOutSubdirectory.BRIDGE_DESCRIPTORS
>>> marker = CollectorOutBridgeDescsMarker.STATUSES
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> fingerprint = "BA44A889E64B93FAA2B114E02C2A279A8555C533" # Serge
>>> filename = collector_422_filename(valid_after, fingerprint)
>>> collector_522_path(subdirectory, marker, valid_after, filename)  # doctest:␣
↪+ELLIPSIS
'bridge-descriptors/statuses/2018/11/19/20181119-150000-BA44...533'
```

Or alternatively for a network-status consensus:

```
>>> subdirectory = CollectorOutSubdirectory.RELAY_DESCRIPTORS
>>> marker = CollectorOutRelayDescsMarker.CONSENSUS
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
```

(continues on next page)

(continued from previous page)

```
>>> filename = collector_431_filename(valid_after)
>>> collector_522_path(subdirectory, marker, valid_after, filename)
'relay-descriptors/consensus/2018/11/19/2018-11-19-15-00-00-consensus'
```

Parameters

- **subdirectory** (*str*) – The subdirectory under the "out" directory to use. Standard values can be found in *CollectorOutSubdirectory*.
- **marker** (*str*) – The marker under the subdirectory to use. Standard values can be found in *CollectorOutRelayDescsMarker* and *CollectorOutBridgeDescsMarker*.
- **valid_after** (*datetime*) – The valid_after time.
- **filename** (*str*) – The filename to use as a `str`, typically created with *collector_422_filename()* for bridge statuses, *collector_431_filename()* for network-status consensuses, or *collector_433_filename()* for network-status votes.

Returns Path for the descriptor as a `str`.

bushel.archive.**collector_522_substructure**(*valid_after*)

Create a path substructure according to §5.2.2 of the [collector-protocol]. This is used for bridge statuses, and network-status consensuses and votes. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_522_substructure(valid_after)
'2018/11/19'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Path substructure as a `str`.

bushel.archive.**collector_533_substructure**(*valid_after*)

Create a substructure according to §5.3.3 of the [collector-protocol]. This is used for microdesc-flavored consensuses and microdescriptors. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_533_substructure(valid_after)
'2018/11'
```

bushel.archive.**collector_534_consensus_path**(*valid_after*)

Create a path according to §5.3.4 of the [collector-protocol] for a **microdesc-flavored** consensus. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_534_consensus_path(valid_after)
'relay-descriptors/microdesc/2018/11/consensus-microdesc/19/2018-11-19-15-00-00-
↪consensus-microdesc'
```

bushel.archive.**collector_534_microdescriptor_path**(*valid_after*, *digest*)

Create a path according to §5.3.4 of the [collector-protocol] for a microdescriptor. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> digest = "00d91cf96321fbd536dd07e297a5e1b7e6961ddd10facdd719716e351453168f"
>>> collector_534_microdescriptor_path(valid_after, digest)
'relay-descriptors/microdesc/2018/11/micro/0/0/
↪00d91cf96321fbd536dd07e297a5e1b7e6961ddd10facdd719716e351453168f'
```

This path in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-256 digests.

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> digest = "00D91CF96321FBD536DD07E297A5E1B7E6961DDD10FACDD719716E351453168F"
>>> collector_534_microdescriptor_path(valid_after, digest)
'relay-descriptors/microdesc/2018/11/micro/0/0/
↪00d91cf96321fbd536dd07e297a5e1b7e6961ddd10facdd719716e351453168f'
```

bushel.archive.**parse_file**(*path*, ***kwargs*)

Parses a descriptor from a file.

Parameters

- **str/bytes** (*content*) – String to construct the descriptor from
- **dict** (*kwargs*) – Additional arguments for `stem.descriptor.Descriptor.parse_file()`.

Returns `stem.descriptor.Descriptor` subclass for the given content, or a *list* of descriptors if **multiple=True** is provided.

bushel.archive.**prepare_annotated_content**(*descriptor*)

Encodes annotations and prepends them to the descriptor bytes for writing to disk.

Parameters **descriptor** (*Descriptor*) – The descriptor to prepare.

Returns `bytes` for the annotated descriptor.

bushel.archive.**valid_after_now**()

Takes a good guess at the valid-after time of the latest consensus. There is an assumption that there is a new consensus every hour and that it is valid from the top of the hour. Different valid-after times are compliant with [dir-spec] however, and so this may be wrong.

Returns A `datetime` for the top of the hour.

CHAPTER 2

Bandwidth Scanner

This module contains stuff related to bandwidth scanners.

2.1 Bandwidth Files

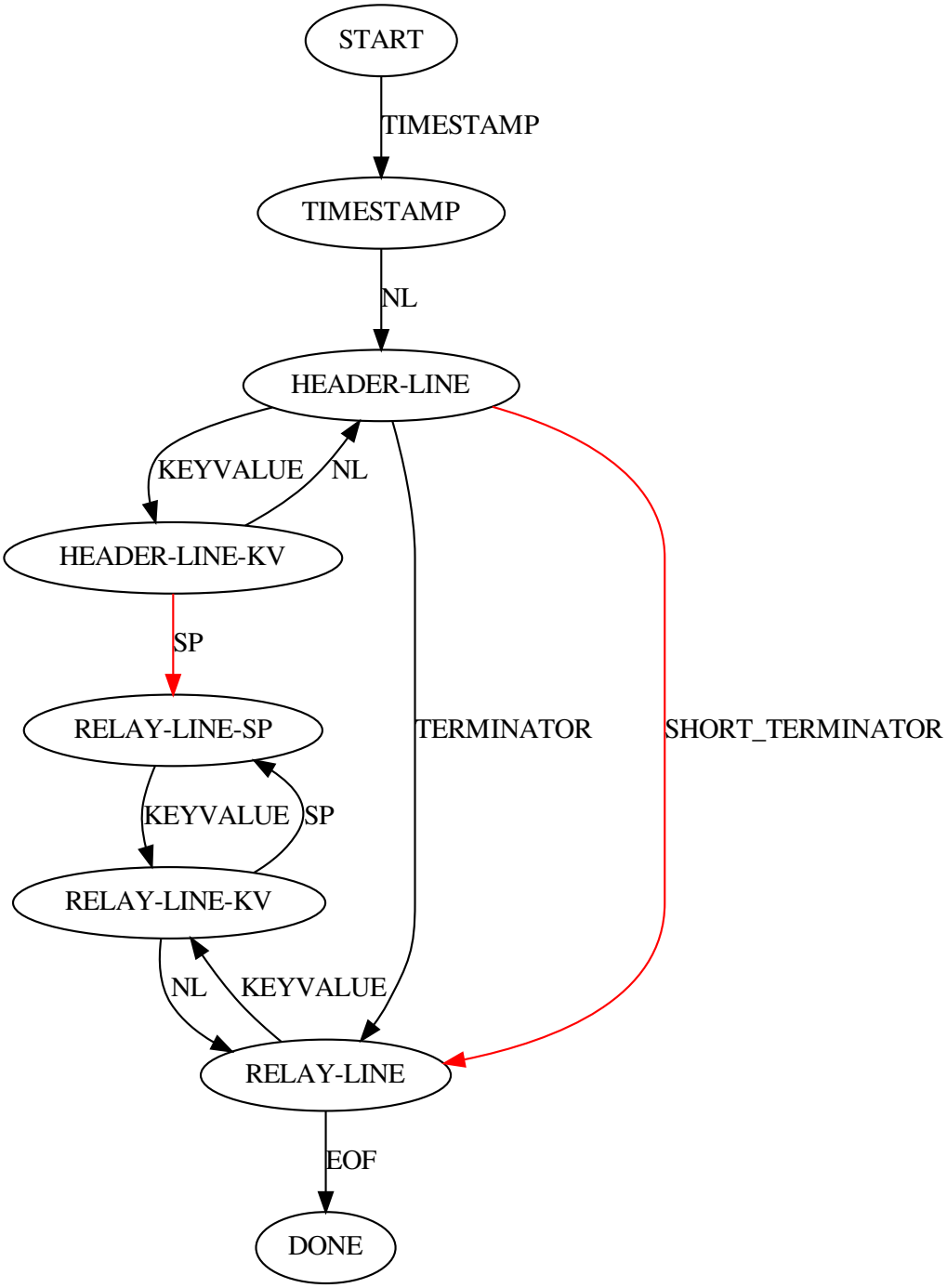
Bandwidth files.

class bushel.bandwidth.file.**BandwidthFileLineError**
Enumeration of forgivable errors that may be encountered during parsing of lines in a bandwidth file.

Name	Description
SHORT_TERMINATOR	A terminator with 4 = instead of 5. https://bugs.torproject.org/28379
NO_TERMINATOR	No terminator present, for pre-1.0.0 compatibility.

class bushel.bandwidth.file.**BandwidthFileLiner**(*allowed_errors=None*)
Parses *BandwidthFileToken* s into BandwidthFileTimestamp, BandwidthFileHeaderLine s and BandwidthFileRelayLine. By default this is a strict implementation of the Tor Bandwidth File Specification version 1.4.0 [bandwidth-file-spec], but this can be relaxed to account for parsing older versions, or for known bugs in Tor implementations.

Lines are produced by processing tokens according to a state machine:



State transitions shown in red would ideally not be needed as they are protocol violations, but implementations of the protocol exist that produce documents requiring these transitions and we need to be bug compatible.

Parameters **allowed_errors** (*list* (BandwidthFileLineError)) – A list of errors that will be considered non-fatal during itemization.

class bushel.bandwidth.file.**BandwidthFileToken**

Variables

- **kind** (*str*) – the kind of token
- **value** (*bytes*) – kind-dependent value
- **line** (*int*) – line number
- **column** (*int*) – column number

CHAPTER 3

CollecTor

3.1 CollecTor Filesystem Protocol

CollecTor Filesystem Protocol.

class bushel.collector.filesystem.**CollecTorIndexCompression**(*extension: str, decompress: Callable[[bytes], bytes]*)

Enumeration of supported compression types for CollecTor indexes.

Name	Description
UNCOMPRESSED	Uncompressed
BZ2	bzip2
XZ	xz
GZ	gzip

Variables **extension** (*str*) – Filename extension with leading dot (".").

class bushel.collector.filesystem.**CollectorOutBridgeDescsMarker**

Enumeration of marker names under the "bridge-descriptors" directory as specified in §5.2 of [collector-protocol].

Name	Description
EXTRA_INFO	Bridge extra-info descriptors (§5.2.1)
SERVER_DESCRIPTOR	Bridge server descriptors (§5.2.1)
STATUS	Bridge statuses (§5.2.2)

class bushel.collector.filesystem.**CollectorOutRelayDescsMarker**

Enumeration of marker names under the "relay-descriptors" directory as specified in §5.3 of [collector-protocol].

Name	Description
CONSENSUS	Network status consensuses (§5.3.2)
EXTRA_INFO	Relay extra-info descriptors (§5.3.2)
SERVER_DESCRIPTOR	Relay server descriptors (§5.3.2)
VOTE	Network status votes (§5.3.2)

class bushel.collector.filesystem.**CollectorOutSubdirectory**
Enumeration of subdirectory names under the "out" directory as specified in §5.0 of [collector-protocol].

Name	Description
BRIDGE_DESCRIPTORS	Bridge descriptors (§5.2)
EXIT_LISTS	Exit lists (§5.1)
RELAY_DESCRIPTORS	Relay descriptors (§5.3)
TORPERF	Torperf and Onionperf (§5.1)
WEBSTATS	Web server access logs (§5.4)

class bushel.collector.filesystem.**CollectorRecentSubdirectory**
Enumeration of subdirectory names under the "recent" directory as specified in §4.0 of [collector-protocol].

Name	Description
BRIDGE_DESCRIPTORS	Bridge descriptors (§4.2)
EXIT_LISTS	Exit lists (§4.1.1)
RELAY_DESCRIPTORS	Relay descriptors (§4.3)
TORPERF	Torperf and Onionperf (§4.1.2)
WEBSTATS	Web server access logs (§4.4)

bushel.collector.filesystem.**collector_422_filename**(*valid_after: datetime.datetime*, *fingerprint: str*) → str
Create a filename for a bridge status according to §4.2.2 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> fingerprint = "BA44A889E64B93FAA2B114E02C2A279A8555C533" # Serge
>>> collector_422_filename(valid_after, fingerprint)
'20181119-150000-BA44A889E64B93FAA2B114E02C2A279A8555C533'
```

Parameters

- **valid_after** (*datetime*) – The valid-after time.
- **fingerprint** (*str*) – The fingerprint of the bridge authority.

Returns Filename as a str.

bushel.collector.filesystem.**collector_431_filename**(*valid_after: datetime.datetime*) → str
Create a filename for a network status consensus according to §4.3.1 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_431_filename(valid_after)
'2018-11-19-15-00-00-consensus'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Filename as a str.

bushel.collector.filesystem.**collector_433_filename**(*valid_after: datetime.datetime*, *v3ident: str*, *digest: str*) → str

Create a filename for a network status vote according to §4.3.3 of the [collector-protocol].

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> v3ident = "D586D18309DED4CD6D57C18FDB97EFA96D330566"  # moria1
>>> digest = "663B503182575D242B9D8A67334365FF8ECB53BB"
>>> collector_433_filename(valid_after, v3ident, digest)  # doctest: +ELLIPSIS
'2018-11-19-15-00-00-vote-D586D18309DED4CD6D57C18FDB97EFA96D330566-663B...3BB'
```

Paths in the Collector File Structure Protocol using this filename expect *upper-case* hex-encoded SHA-1 digests.

```
>>> v3ident = "d586d18309ded4cd6d57c18fdb97efa96d330566"  # Lower case gets␣
↪corrected
>>> digest = "663b503182575d242b9d8a67334365ff8ecb53bb"  # Lower case gets␣
↪corrected
>>> collector_433_filename(valid_after, v3ident, digest)  # doctest: +ELLIPSIS
'2018-11-19-15-00-00-vote-D586D18309DED4CD6D57C18FDB97EFA96D330566-663B...3BB'
```

Parameters

- **valid_after** (*datetime*) – The valid-after time.
- **v3ident** (*str*) – The v3ident of the directory authority.
- **digest** (*str*) – The digest of the vote.

Returns Filename as a `str`.

bushel.collector.filesystem.**collector_434_filename**(*valid_after: datetime.datetime*) → str

Create a filename for a microdesc-flavoured network status consensus according to §4.3.4 of the [collector-protocol]. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_434_filename(valid_after)
'2018-11-19-15-00-00-consensus-microdesc'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Filename as a `str`.

bushel.collector.filesystem.**collector_521_path**(*subdirectory: bushel.collector.filesystem.CollectorOutSubdirectory*, *marker: Union[bushel.collector.filesystem.CollectorOutRelayDes bushel.collector.filesystem.CollectorOutBridgeDescsMarker]*, *published: datetime.datetime*, *digest: str*) → str

Create a path according to §5.2.1 of the [collector-protocol]. This is used for server-descriptors and extra-info descriptors for both relays and bridges. For example:

```
>>> subdirectory = CollectorOutSubdirectory.RELAY_DESCRIPTORS
>>> marker = CollectorOutRelayDescsMarker.SERVER_DESCRIPTOR
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> collector_521_path(subdirectory, marker, published, digest)  # doctest:␣
↪+ELLIPSIS
'relay-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

Paths in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-1 digests.

```
>>> digest = "A94A07B201598D847105AE5FCD5BC3AB10124389" # Upper case gets␣
↪corrected
>>> collector_521_path(subdirectory, marker, published, digest)  # doctest:␣
↪+ELLIPSIS
'relay-descriptors/server-descriptor/2018/11/a/9/a94a...389'
```

Parameters

- **subdirectory** (*str*) – The subdirectory under the "out" directory to use. Standard values can be found in *CollectorOutSubdirectory*.
- **marker** (*str*) – The marker under the subdirectory to use. Standard values can be found in *CollectorOutRelayDescsMarker* and *CollectorOutBridgeDescsMarker*.
- **published** (*datetime*) – The published time.
- **digest** (*str*) – The hex-encoded SHA-1 digest for the descriptor. The case will automatically be fixed to lower-case.

Returns Path for the descriptor as a `str`.

bushel.collector.filesystem.**collector_521_substructure**(*published: datetime.datetime*, *digest: str*) → str

Create a path substructure according to §5.2.1 of the [collector-protocol]. This is used for server-descriptors and extra-info descriptors for both relays and bridges. For example:

```
>>> published = datetime.datetime(2018, 11, 19, 9, 17, 56)
>>> digest = "a94a07b201598d847105ae5fcd5bc3ab10124389"
>>> collector_521_substructure(published, digest)
'2018/11/a/9'
```

Paths in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-1 digests.

```
>>> digest = "A94A07B201598D847105AE5FCD5BC3AB10124389" # Upper case gets␣
↪corrected
>>> collector_521_substructure(published, digest)
'2018/11/a/9'
```

Parameters

- **published** (*datetime*) – The published time.
- **digest** (*str*) – The hex-encoded SHA-1 digest for the descriptor. The case will automatically be fixed to lower-case.

Returns Path substructure as a `str`.

bushel.collector.filesystem.**collector_522_path**(*subdirectory: bushel.collector.filesystem.CollectorOutSubdirectory*, *marker: Union[bushel.collector.filesystem.CollectorOutRelayDes bushel.collector.filesystem.CollectorOutBridgeDescsMarker]*, *valid_after: datetime.datetime*, *filename: str*) → str

Create a path according to §5.2.2 of the [collector-protocol]. This is used for bridge statuses, and network-status

consensuses (both ns- and microdesc- flavors) and votes. For a bridge status for example:

```
>>> subdirectory = CollectorOutSubdirectory.BRIDGE_DESCRIPTORS
>>> marker = CollectorOutBridgeDescsMarker.STATUSES
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> fingerprint = "BA44A889E64B93FAA2B114E02C2A279A8555C533" # Serge
>>> filename = collector_422_filename(valid_after, fingerprint)
>>> collector_522_path(subdirectory, marker, valid_after, filename)  # doctest:␣
↪+ELLIPSIS
'bridge-descriptors/statuses/2018/11/19/20181119-150000-BA44...533'
```

Or alternatively for a network-status consensus:

```
>>> subdirectory = CollectorOutSubdirectory.RELAY_DESCRIPTORS
>>> marker = CollectorOutRelayDescsMarker.CONSENSUS
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> filename = collector_431_filename(valid_after)
>>> collector_522_path(subdirectory, marker, valid_after, filename)
'relay-descriptors/consensus/2018/11/19/2018-11-19-15-00-00-consensus'
```

Parameters

- **subdirectory** (*str*) – The subdirectory under the "out" directory to use. Standard values can be found in *CollectorOutSubdirectory*.
- **marker** (*str*) – The marker under the subdirectory to use. Standard values can be found in *CollectorOutRelayDescsMarker* and *CollectorOutBridgeDescsMarker*.
- **valid_after** (*datetime*) – The valid_after time.
- **filename** (*str*) – The filename to use as a `str`, typically created with *collector_422_filename()* for bridge statuses, *collector_431_filename()* for network-status consensuses, or *collector_433_filename()* for network-status votes.

Returns Path for the descriptor as a `str`.

bushel.collector.filesystem.**collector_522_substructure**(*valid_after: datetime.datetime*) → str

Create a path substructure according to §5.2.2 of the [collector-protocol]. This is used for bridge statuses, and network-status consensuses and votes. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_522_substructure(valid_after)
'2018/11/19'
```

Parameters **valid_after** (*datetime*) – The valid-after time.

Returns Path substructure as a `str`.

bushel.collector.filesystem.**collector_533_substructure**(*valid_after: datetime.datetime*) → str

Create a substructure according to §5.3.3 of the [collector-protocol]. This is used for microdesc-flavored consensuses and microdescriptors. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_533_substructure(valid_after)
'2018/11'
```

bushel.collector.filesystem.**collector_534_consensus_path**(*valid_after*)

Create a path according to §5.3.4 of the [collector-protocol] for a **microdesc-flavored** consensus. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> collector_534_consensus_path(valid_after)  # doctest: +ELLIPSIS
'relay-descriptors/microdesc/2018/11/consensus-microdesc/19/2018-11-1...sc'
```

bushel.collector.filesystem.**collector_534_microdescriptor_path**(*valid_after: datetime.datetime*, *digest: str*) → str

Create a path according to §5.3.4 of the [collector-protocol] for a microdescriptor. For example:

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> digest = "00d91cf96321fbd536dd07e297a5e1b7e6961ddd10facdd719716e351453168f"
>>> collector_534_microdescriptor_path(valid_after, digest)  # doctest: +ELLIPSIS
'relay-descriptors/microdesc/2018/11/micro/0/0/00d...e351453168f'
```

This path in the Collector File Structure Protocol using this substructure expect *lower-case* hex-encoded SHA-256 digests.

```
>>> valid_after = datetime.datetime(2018, 11, 19, 15)
>>> digest = "00D91CF96321FBD536DD07E297A5E1B7E6961DDD10FACDD719716E351453168F"
>>> collector_534_microdescriptor_path(valid_after, digest)  # doctest: +ELLIPSIS
'relay-descriptors/microdesc/2018/11/micro/0/0/00d...e351453168f'
```

bushel.collector.filesystem.**collector_index_path**(*compression: bushel.collector.filesystem.CollecTorIndexCompression*) → str

Create a path to the CollecTor index file, using the specified compression algorithm.

Parameters **compression** (CollecTorIndexCompression) – Compression algorithm to use.

3.2 CollecTor Remotes

Remote CollecTor instance interaction.

This module provides tools for interacting with remote CollecTor instances, such as those run by Tor Metrics or 3rd-party public or private CollecTor instances.

bushel.collector.remote.**DEFAULT_COLLECTOR_HOST**

The default CollecTor host to use when none is specified, currently *collector.torproject.org* although this is subject to change. It will be set to the currently recommended public Tor Metrics instance.

bushel.collector.remote.**DEFAULT_INDEX_COMPRESSION**

The default compression algorithm used with CollecTor indexes. This is currently set to *xz* although is subject to change in line with any recommendations from Tor Metrics.

class bushel.collector.remote.**CollecTorRemote**(*host: Optional[str] = None*, ***, *https: bool = True*)

A remote CollecTor instance. Methods are provided for querying the data available on the remote instance, as well as retrieving data from the remote instance.

Parameters

- **host** (*str*) – The FQDN of the CollecTor instance. If None, then the *DEFAULT_COLLECTOR_HOST* is used.
- **https** (*bool*) – Whether HTTPS should be used. This defaults to *True*.

get_index(*compression: Optional[bushel.collector.filesystem.CollecTorIndexCompression]*) → bushel.collector.index.CollecTorIndex

Fetch the index from the CollecTor instance, optionally specifying the compression algorithm to use. This function will return an object that contains the (decompressed if necessary) and parsed index.

Parameters **compression** (CollecTorIndexCompression) – Compression algorithm to use. If *None*, the default specified in *DEFAULT_INDEX_COMPRESSION* will be used.

Return type CollecTorIndex

get_raw_by_path(*path: str*) → bytes

Fetch the raw bytes of a file from a CollecTor instance.

Parameters **path** (*str*) – CollecTor path with no leading slash (/).

Return type bytes

Returns Raw bytes of the reply, which may be compressed depending on the requested path.

bushel.collector.remote.**get_index**(*host: Optional[str] = None*, *compression: Optional[bushel.collector.filesystem.CollecTorIndexCompression] = None*, ***, *https: bool = True*) → bushel.collector.index.CollecTorIndex

Convenience function for `CollecTorRemote(host, https=https).get_index(compression)`.

See also:

CollecTorRemote.get_index()

3.3 CollecTor Indexes

CHAPTER 4

Tor Directory Protocol

4.1 Directory Documents

The *bushel.directory.document* module provides base classes and utility methods for handling documents that implement the Tor directory protocol version 3 meta format (§1.2 [dir-spec]).

For specific document types, see:

4.1.1 Detached Signatures

class bushel.directory.detached_signature.**DetachedSignature**(*raw_content*)

Detached signature documents are used as part of the consensus process for the Tor directory protocol version 3 (§3.10 [dir-spec]). Once an authority has computed and signed a consensus network status, it should send its detached signature to each other authority in an HTTP POST request. All of the detached signatures it knows for consensus status should be available at:

`http://<hostname>/tor/status-vote/next/consensus-signatures.z`

Assuming full connectivity, every authority should compute and sign the same consensus including any flavors in each period. Therefore, it isn't necessary to download the consensus or any flavors of it computed by each authority; instead, the authorities only push/fetch each others' signatures.

These documents are interesting for Tor Metrics as they allow detection of new consensus flavors automatically, allowing them to be archived as soon as they are available even if we are not yet able to parse them.

Variables

- **consensus_digest** (*str*) – digest of the consensus
- **valid_after** (*datetime*) – the valid-after time
- **fresh_until** (*datetime*) – the fresh-until time
- **valid_until** (*datetime*) – the valid-until time

- **additional_digests** (*list (DetachedSignatureAdditionalDigest)*) – additional digests
- **additional_signatures** (*list (DetachedSignatureAdditionalSignature)*) – additional signatures
- **direcory_signatures** (*list (NetworkStatusConsensusDirectorySignature)*) – directory signatures

class bushel.directory.detached_signature.**DetachedSignatureAdditionalDigest**
Additional signatures as found in *DetachedSignature* s, defined in the Tor directory protocol version 3 ([dir-spec] §3.10).

Variables

- **flavor** (*str*) – flavor of the additional consensus
- **algname** (*str*) – name of algorithm used for the digest
- **digest** (*str*) – the digest of the document as signed

class bushel.directory.detached_signature.**DetachedSignatureAdditionalSignature**
Additional signatures as found in *DetachedSignature* s, defined in the Tor directory protocol version 3 ([dir-spec] §3.10).

Variables

- **flavor** (*str*) – flavor of the additional consensus
- **algname** (*str*) – name of algorithm used for the digest
- **identity** (*str*) – hex-encoded digest of the authority identity key of the signing authority
- **signing_key_digest** (*str*) – hex-encoded digest of the current authority signing key of the signing authority
- **signature** (*bytes*) – RSA signature of the OAEP+-padded SHA256 digest of the additional consensus

4.1.2 Network Statuses

class bushel.directory.network_status.**NetworkStatusConsensus**(*raw_content*)

class bushel.directory.network_status.**NetworkStatusConsensusDirectorySignature**
Directory signatures as found in *NetworkStatusConsensus*, defined in the Tor directory protocol version 3 ([dir-spec] §3.4.1).

For the signature, we take the hash through the _space_ after `directory-signature`, not the newline: this ensures that all authorities sign the same thing.

Variables

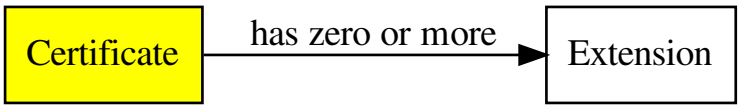
- **algorithm** (*str*) – one of "sha1" or "sha256", or *None* if this was not present
- **identity** (*str*) – hex-encoded digest of the authority identity key of the signing authority
- **signing_key_digest** (*str*) – hex-encoded digest of the current authority signing key of the signing authority
- **signature** (*bytes*) – signature of the status document, with the initial item "network-status-version", and the signature item "directory-signature", using the signing key

4.1.3 Server Descriptors

4.1.4 Extra Info Descriptors

class bushel.directory.document.**DirectoryCertificate**(*raw_content*)

A Tor Ed25519 certificate as specified by [cert-spec]. It is not the only certificate format that Tor uses. Typically these are found as the data contained within *DirectoryDocumentObject* s.



Parameters **raw_content** (*bytes*) – raw certificate contents

Variables

- **data** (*bytes*) – raw certificate contents
- **version** (*int*) – version of the certificate format (currently always 1)
- **cert_type** (*int*) – type of certificate
- **expiration_date** (*datetime*) – expiration date of certificate
- **cert_key_type** (*int*) – type of certified key
- **certified_key** (*bytes*) – an Ed25519 public key if cert_key_type is 1, or a SHA256 hash of some other key type depending on the value of cert_key_type
- **n_extensions** (*int*) – declared number of extensions
- **extensions** (*list (DirectoryCertificateExtension)*) – parsed extensions
- **signature** (*bytes*) – certificate signature

is_valid()

Checks that the certificate is valid. This is the counterpart to *verify()* that checks that the certificate data conforms to the specification. The two checks performed are:

- expiration date is not passed
- there are no extensions that affect validation that we do not understand

---

Note: In the Tor Metrics use case, we need to check that certificates were valid at the time they were expected to be valid, but the current API does not support this.

---

parse()

Parses the certificate to make the fields available via instance attributes. This does not validate or verify the certificate, but must be called before making calls to *is_valid()* or *verify()*.

verify(*verify_key_data=None*)

Verify the certificate using the verification key. Optionally provide key material, otherwise the key found in the "signed-with-ed25519-key" (type 4) extension will be used.

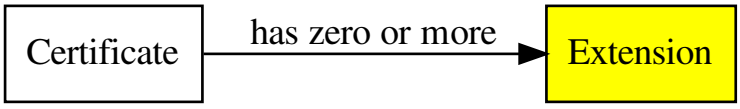
This only verifies the signature. To validate the certificate data the seperate *DirectoryCertificate.is_valid()* method must be used.

Warning: This verifies the raw data that the object was initialized with, the fields may have been played with since parsing and the parser may also have unknown bugs.

Parameters **verify_key_data** (*bytes*) – an Ed25519 verification key

class bushel.directory.document.**DirectoryCertificateExtension**

A Tor Ed25519 certificate extension as specified by [cert-spec].



Variables

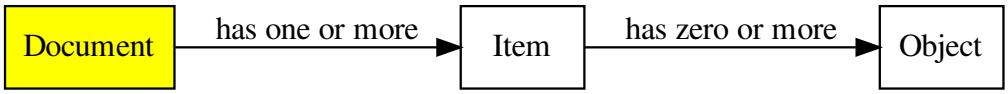
- **type** (*int*) – extension type
- **flags** (*int*) – extension flags
- **data** (*bytes*) – extension data

See also:

These will be found in *DirectoryCertificate* s.

class bushel.directory.document.**DirectoryDocument**(*raw_content*)

A directory document as described in the Tor directory protocol meta format (§1.2 [dir-spec]).



Parameters **raw_content** (*bytes*) – raw document contents

tokenize()

Tokenizes the document using the following tokens:

Kind	Matches on	Value
END	`"-----END " Keyword "-----"`	Keyword
BEGIN	`"-----BEGIN " Keyword "-----"`	Keyword
NL	The ascii LF character (hex value 0x0a)	Raw data
PRINTABLE	Printing, non-whitespace, UTF-8	Raw data
WS	Space or tab	Raw data
MISMATCH	Anything else (likely binary nonsense)	Raw data

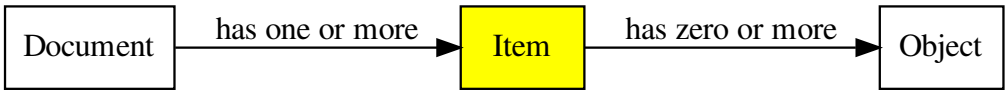
Note that these tokens do not match the non-terminals exactly as they are specified in the Tor directory protocol meta format. In particular, the PRINTABLE token is used for both keywords and arguments (and object data). It is up to whatever is processing these tokens to decide if something is valid keyword or argument.

```
>>> document_bytes = b'''super-keyword 3
... onion-magic
... -----BEGIN ONION MAGIC-----
... AQQABp6MAT7yJjlcuWLDbr8A5J8YgyDh5SPYkLpj7fmcBaFbKekjAQAgBADKnR/C
... -----END ONION MAGIC-----
... '''
>>> for token in DirectoryDocument(document_bytes).tokenize():
...     print(token) # doctest: +ELLIPSIS
DirectoryDocumentToken(kind='PRINTABLE', value='super-keyword', line=1,␣
↪column=0)
DirectoryDocumentToken(kind='WS', value=' ', line=1, column=13)
DirectoryDocumentToken(kind='PRINTABLE', value='3', line=1, column=14)
DirectoryDocumentToken(kind='NL', value='\n', line=1, column=15)
DirectoryDocumentToken(kind='PRINTABLE', value='onion-magic', line=2,␣
↪column=0)
DirectoryDocumentToken(kind='NL', value='\n', line=2, column=11)
DirectoryDocumentToken(kind='BEGIN', value='ONION MAGIC', line=3, column=0)
DirectoryDocumentToken(kind='PRINTABLE', value='AQQ...DKnR/C', line=4,␣
↪column=0)
DirectoryDocumentToken(kind='NL', value='\n', line=4, column=64)
DirectoryDocumentToken(kind='END', value='ONION MAGIC', line=5, column=0)
DirectoryDocumentToken(kind='EOF', value=None, line=6, column=0)
```

Returns iterator for *DirectoryDocumentToken*

class bushel.directory.document.**DirectoryDocumentItem**(*keyword*, *arguments*, *objects*, *errors*)

A directory document item as described in the Tor directory protocol meta format (§1.2 [dir-spec]).



Parameters

- **keyword** (*bytes*) – the item keyword
- **arguments** (*list (bytes)*) – list of item arguments

- **objects** (*list(tuple(bytes,bytes))*) – list of item objects as tuples of (object keyword, decoded object data)
- **errors** (*list(DirectoryDocumentItemError)*) – list of errors found during item parsing

Variables

- **keyword** (*bytes*) – the item keyword
- **arguments** (*list(bytes)*) – list of item arguments
- **objects** (*list(tuple(bytes,bytes))*) – list of item objects as tuples of (object keyword, decoded object data)
- **errors** (*list(DirectoryDocumentItemError)*) – list of errors found during item parsing

class bushel.directory.document.**DirectoryDocumentItemError**

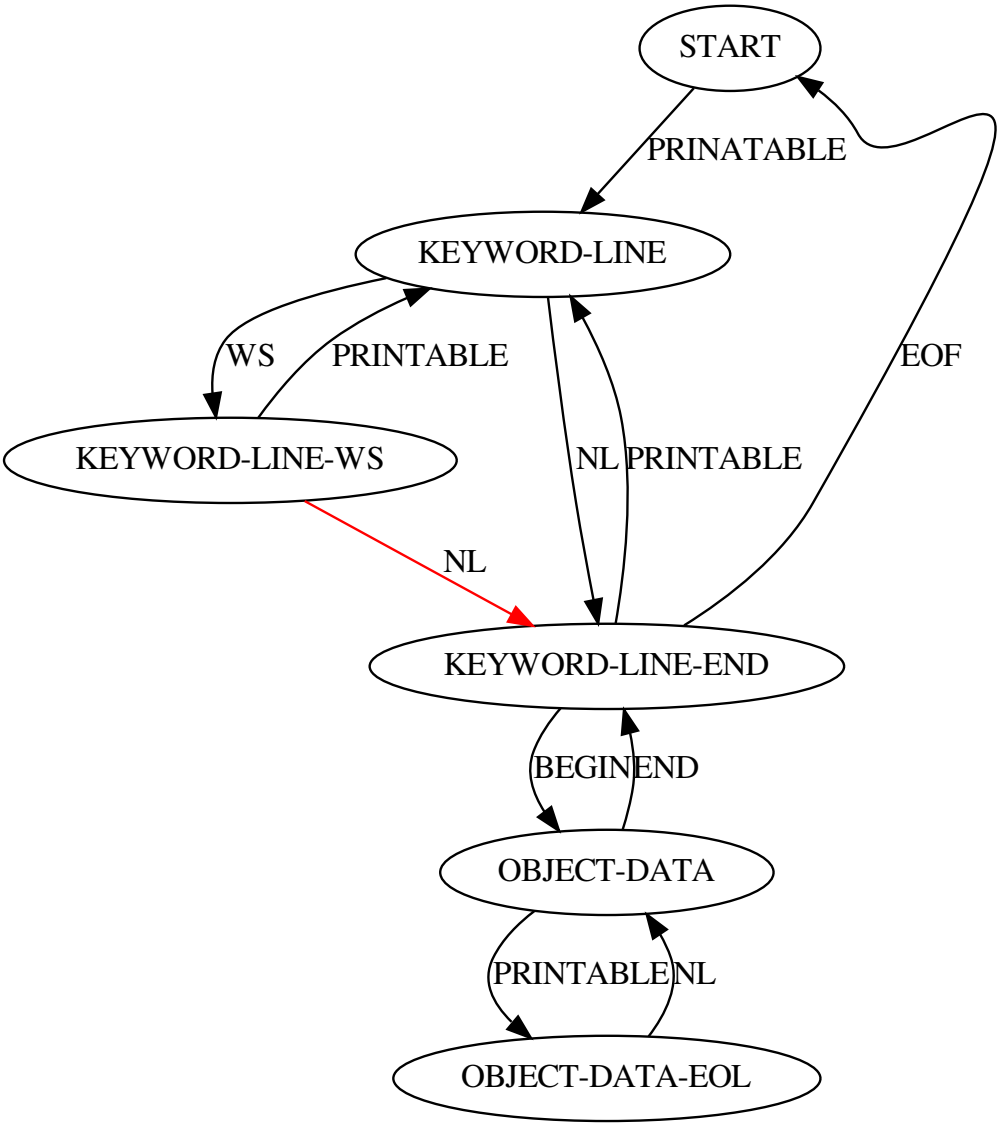
Enumeration of forgivable errors that may be encountered during itemization of a directory document.

Name	Description
TRAILING_WHITESPACE	Trailing whitespace on KeywordLines https://bugs.torproject.org/30105

class bushel.directory.document.**DirectoryDocumentItemizer**(*allowed_errors=None*)

Parses *DirectoryDocumentToken* s into *DirectoryDocumentItem* s. By default this is a strict implementation of the Tor directory protocol meta format (§1.2 [dir-spec]), but this can be relaxed to account for implementation bugs in known Tor implementations.

Items are produced by processing tokens according to a state machine:



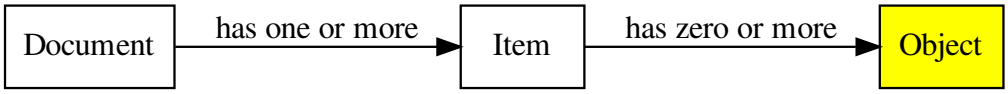
State transitions shown in red would ideally not be needed as they are protocol violations, but implementations of the protocol exist that produce documents requiring these transitions and we need to be bug compatible.

> **Warning:** All printable strings are treated equally right now, so we're not testing for keywords being the restricted set, nor are we decoding object data yet.

Parameters **allowed_errors** (*list(DirectoryDocumentItemError)*) – A list of errors that will be considered non-fatal during itemization.

class bushel.directory.document.**DirectoryDocumentObject**

A directory document item as described in the Tor directory protocol meta format (§1.2 [dir-spec]).



Variables

- **keyword** (*bytes*) – object keyword
- **data** (*bytes*) – decoded object data

class bushel.directory.document.**DirectoryDocumentToken**

Variables

- **kind** (*str*) – the kind of token
- **value** (*bytes*) – kind-dependent value
- **line** (*int*) – line number
- **column** (*int*) – column number

bushel.directory.document.**decode_object_data**(*lines*)

Decodes the base64 encoded data found within directory document objects.

Parameters **lines** (*list(str)*) – the lines as found in a directory document object, not including newlines or the begin/end lines

Returns the decoded data

Return type bytes

bushel.directory.document.**encode_object_data**(*data*)

Encodes bytes using base64 and wraps the lines at 64 charachters.

Parameters **data** (*bytes*) – the data to be encoded

Returns the line-wrapped base64 encoded data as a list of strings, one string per line

Return type list(str)

bushel.directory.document.**parse_timestamp**(*item*, *argindex=0*)

Parses a timestamp from a directory document's item using the common format from [dir-spec]. This format is not defined explicitly but is used with many keywords including `valid-after`, `fresh-until`, and `valid-until`.

Note: Due to the way the tokenizer works, timestamps are parsed as two arguments split by whitespace. This function takes this into account when parsing the timestamp.

Most items will have the timestamp as the first argument on the keyword line. At the time of writing, there are no keywords defined that expect timestamps at other indexes. Should this be required though, *argindex* may be used to parse a timestamp from a later argument.

Parameters

- **item** (DirectoryDocumentItem) – the directory document item

- **argindex** (*int*) – zero-indexed index of date portion of timestamp, the time portion is expected in `argindex+1`

Returns the parsed timestamp

Return type datetime

4.2 Directory Client

4.3 Directory Voting

An implementation of the voting process used in the Tor directory protocol, version 3 [dir-spec].

`bushel.directory.voting.`**`valid_after_now_guess`**`()`

Takes a good guess at the valid-after time of the latest consensus. There is an assumption that there is a new consensus every hour and that it is valid from the top of the hour. Different valid-after times are compliant with the protocol however, and so this may be wrong.

The voting timeline is described in §1.4 of the Tor directory protocol, version 3 ([dir-spec]).

Returns The start of the current hour in UTC.

Return type datetime

CHAPTER 5

Directory Downloader

class bushel.downloader.**DirectoryDownloader**(*initial_consensus=None*, *directory_cache_mode=None*, *max_concurrency=9*)

The *DirectoryDownloader* provides an asyncio-compatible wrapper around the stem DescriptorDownloader, with two modes of operation:

- Directory Cache ([dir-spec] §4)
- Client ([dir-spec] §5)

The DirectoryDownloader will not initiate downloads on its own intiative. It must be driven to perform downloads through the methods provided.

Note: As a valid consensus is required to implement parts of the functionality, the latest consensus is cached internally. This cached consensus should not be relied upon by external code. The cached consensus will never be served as a response to a request for a consensus.

---

directory_authorities()

Returns a list containing either a DirPort or an ORPort for each of the directory authorities.

directory_caches(*extra_info=False*)

Returns a list containing either a DirPort or an ORPort for each of the directory caches known from the latest consensus. If no consensus is known, this will return authorities() instead.

Parameters **extra_info** (*bool*) – Whether the list returned should contain only directory caches that cache extra-info descriptors.

relay_extra_info_descriptors(*digests*, *published_hint=None*)

Retrieves multiple extra-info descriptors from directory servers.

Parameters

- **digests** (*list (str)*) – Hex-encoded digests for the descriptors.

- **published_hint** (*datetime*) – Provides a hint on the published time. Currently this is unused, but is accepted for compatibility with other directory sources. In the future this may be used to avoid attempts to download descriptors that it is likely are long gone.

Returns A list of stem.descriptor.extrainfo_descriptor.RelayExtraInfoDescriptor.

relay_microdescriptors(*microdescriptor_hashes*, *valid_after_hint=None*)
Retrieves multiple server descriptors from directory servers.

Parameters

- **hashes** (*list(str)*) – base64-encoded hashes for the microdescriptors.
- **valid_after_hint** (*datetime*) – Provides a hint on the valid_after time. Currently this is unused, but is accepted for compatibility with other directory sources. In the future this may be used to avoid attempts to download descriptors that it is likely are long gone.

Returns A list of stem.descriptor.microdescriptor.Microdescriptor.

relay_server_descriptors(*digests*, *published_hint=None*)
Retrieves multiple server descriptors from directory servers.

Parameters

- **digests** (*list(str)*) – Hex-encoded digests for the descriptors.
- **published_hint** (*datetime*) – Provides a hint on the published time. Currently this is unused, but is accepted for compatibility with other directory sources. In the future this may be used to avoid attempts to download descriptors that it is likely are long gone.

Returns A list of stem.descriptor.server_descriptor.RelayDescriptor.

bushel.downloader.**relay_extra_info_descriptors_query_path**(*digests*)
Generates a query path to request extra-info descriptors by digests from a directory server. For example:

```
>>> digests = ["A94A07B201598D847105AE5FCD5BC3AB10124389",
...            "B3897498732339479587383ABEF4893BD4895A8"]
>>> relay_extra_info_descriptors_query_path(digests)  # doctest: +ELLIPSIS
'/tor/extra/d/A94A07B201598D847105...24389+B3897498732339479587...95A8'
```

These query paths are defined in appendix B of [dir-spec]. By convention, these URLs use upper-case hex-encoded SHA-1 digests and so this function will ensure that digests are upper-case. Directory server implementations should not rely on this behaviour.

Parameters **digests** (*list(str)*) – The hex-encoded SHA-1 digests for the descriptors.

Returns Query path as a str.

bushel.downloader.**relay_microdescriptors_query_path**(*microdescriptor_hashes*)
Generates a query path to request microdescriptors by their hashes from a directory server. For example:

```
>>> microdescriptor_hashes = ["Z62HG1C9PLIVs8jLi1guO48rzPdcq6tFTLi5s27Zy4U",
...                           "FkiLuQJe/Gqp4xsHfheh+G42TSJ77AarHOGrjazj0Q0"]
>>> relay_microdescriptors_query_path(microdescriptor_hashes)  # doctest:␣
↪+ELLIPSIS
'/tor/micro/d/Z62HG1C9PLIVs8jL...Li5s27Zy4U-FkiLuQJe/Gqp4xsHf...rjazj0Q0'
```

These query paths are defined in appendix B of [dir-spec].

Parameters **digests** (*list(str)*) – The base64-encoded hashes for the descriptors.

Returns Query path as a str.

bushel.downloader.**relay_server_descriptors_query_path**(*digests*)

Generates a query path to request server descriptors by digests from a directory server. For example:

```
>>> digests = ["A94A07B201598D847105AE5FCD5BC3AB10124389",
...            "B389749873233947958793 83ABEF4893BD4895A8"]
>>> relay_server_descriptors_query_path(digests)  # doctest: +ELLIPSIS
'/tor/server/d/A94A07B201598D847105...24389+B389749873233947958 7...95A8'
```

These query paths are defined in appendix B of [dir-spec]. By convention, these URLs use upper-case hex-encoded SHA-1 digests and so this function will ensure that digests are upper-case. Directory server implementations should not rely on this behaviour.

Parameters **digests** (*list(str)*) – The hex-encoded SHA-1 digests for the descriptors.

Returns Query path as a str.

CHAPTER 6

Monitoring

For bundled plugins for Nagios see:

6.1 check_collector

6.1.1 Check a CollecTor instance for operational issues

Manual section 1

SYNOPSIS

check_collector hostname [module]

DESCRIPTION

Checks a CollecTor instance to ensure that the files it is serving are fresh.

hostname The hostname of the CollecTor instance. There are no defaults to avoid implicit misconfiguration accidents. Example: “collector.torproject.org”.

module The module to test. If not specified, the script will run through all available modules to make sure they are working. When configured for use with Nagios or compatible software this should be set to one of: “index”, “relaydescs”, “bridgedescs”, “exitlists”.

EXAMPLES

Run all the checks on the command line to ensure the installation is working or to perform a one-off test of the Tor Metrics CollecTor instance:

```
check_collector collector.torproject.org
```

Check the Tor Metrics CollecTor instance to see that the relaydescs module has been running:

```
check_collector collector.torproject.org relaydescs
```

BUGS

- bridgedescs module does not check network status document timestamps as the timestamp format is different

Please report any bugs found to: https://github.com/irl/bushel/issues.

AUTHORS

check_collector is part of bushel, a Python library and application supporting parts of Tor Metrics.

check_collector and this man page were written by Iain Learmonth <irl@torproject.org>.

6.2 Monitoring Helpers

This module contains tools for creating plugins for monitoring applications that are compatible with Nagios plugins, such as Nagios or Icinga.

bushel.monitoring.**NagiosStatusCode**
: Derived type to represent the exit code of a Nagios plugin.

bushel.monitoring.**NagiosResponse**
: Alias for `typing.Tuple[NagiosStatusCode, str]` to represent the exit code and message for a Nagios plugin.

bushel.monitoring.**OK**

bushel.monitoring.**WARNING**

bushel.monitoring.**CRITICAL**

bushel.monitoring.**UNKNOWN**
: Standard Nagios plugin return codes. These constants are instances of *NagiosStatusCode*.

CHAPTER 7

Plugin API

The following documents a draft API to be implemented by plugins. These functions will be called by the reference checker. While plugins may keep state internally, it is expected that any state they do keep is not required to be persistent.

DocumentIdentifier(doctype, subject, datetime, digests):
Represents a document that is expected to exist.

Attributes:

doctype
The `type` of the document.

subject
The subject of the document. This is usually a string containing an opaque identifier. Examples include the fingerprint of a relay for a server descriptor, or the hostname of an OnionPerf vantage point.

datetime
A `datetime` related to the document. The exact meaning of this will be document dependent. Example include the published time for a server descriptor, or the valid-after time for a network status consensus.

digests
A `dict` containing mappings of `DigestHash` to `tuple`s. Each tuple contains a :py:class:`str` representation of the digest and a `stem.descriptor.DigestEncoding`.

class ExamplePlugin

expectations()

Returns A `list` of `DocumentIdentifier` for documents that are expected to be available for fetching.

fetch(*docid*)
Fetches a document from a remote location.

Parameters **docid** (*DocumentIdentifier*) – Identifier for the document to be fetched.

parse(*document*)

Parses a retrieved document for any documents that are referenced and should be fetched.

Parameters **document** (*DocumentIdentifier*) – A retrieved document.

Returns A `list` of `DocumentIdentifier` for documents that are expected to be available for fetching.

CHAPTER 8

References

The requirements and initial design specification for the Tor directory archive can be found in the 2019 Technical Report [modern-collector].

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[bandwidth-file-spec] Tor Bandwidth File Format. https://spec.torproject.org/bandwidth-file-spec

[cert-spec] Ed25519 certificates in Tor. https://spec.torproject.org/cert-spec

[collector-protocol] Protocol of CollecTor's File Structure. https://spec.torproject.org/collector-protocol

[dir-spec] Tor directory protocol, version 3. https://spec.torproject.org/dir-spec

[modern-collector] Iain R. Learmonth and Karsten Loesing. Towards modernising data collection and archive for the Tor network. Technical Report 2018-12-001, The Tor Project, December 2018. https://research.torproject.org/techreports/modern-collector-2018-12-19.pdf

Python Module Index

b

Index

D

E

F

G

I

N

O

P

R

S

T

U

V

W