```
      "650" SP "STREAM" SP StreamID SP StreamStatus SP CircuitID SP
Target
          [SP "REASON=" Reason [ SP "REMOTE_REASON=" Reason ]]
          [SP "SOURCE=" Source] [ SP "SOURCE_ADDR=" Address ":" Port ]
          [SP "PURPOSE=" Purpose] [SP "SOCKS_USERNAME="
EscapedUsername]
          [SP "SOCKS_PASSWORD=" EscapedPassword]
          [SP "CLIENT_PROTOCOL=" ClientProtocol] [SP "NYM_EPOCH="
NymEpoch]
          [SP "SESSION_GROUP=" SessionGroup] [SP "ISO_FIELDS="
IsoFields]
          CRLF

      StreamStatus =
               "NEW"          / ; New request to connect
               "NEWRESOLVE"   / ; New request to resolve an address
               "REMAP"        / ; Address re-mapped to another
               "SENTCONNECT"  / ; Sent a connect message along a
circuit
               "SENTRESOLVE"  / ; Sent a resolve message along a
circuit
               "SUCCEEDED"    / ; Received a reply; stream established
               "FAILED"       / ; Stream failed and not retriable
               "CLOSED"       / ; Stream closed
               "DETACHED"     / ; Detached from circuit; still
retriable
               "CONTROLLER_WAIT"  ; Waiting for controller to use
ATTACHSTREAM
                                  ; (new in 0.4.5.1-alpha)
               "XOFF_SENT"  ; XOFF has been sent for this stream
                            ; (new in 0.4.7.5-alpha)
               "XOFF_RECV"  ; XOFF has been received for this stream
                            ; (new in 0.4.7.5-alpha)
               "XON_SENT"  ; XON has been sent for this stream
                           ; (new in 0.4.7.5-alpha)
               "XON_RECV"  ; XON has been received for this stream
                           ; (new in 0.4.7.5-alpha)

      Target = TargetAddress ":" Port
      Port = an integer from 0 to 65535 inclusive
      TargetAddress = Address / "(Tor_internal)"

      EscapedUsername = QuotedString
      EscapedPassword = QuotedString

      ClientProtocol =
               "SOCKS4"      /
               "SOCKS5"      /
               "TRANS"       /
               "NATD"        /
               "DNS"         /
               "HTTPCONNECT" /
               "UNKNOWN"
```

Specification

Pluggable Transport proxies follow the following workflow throughout their lifespan.

```
     1) Parent process sets the required environment values (3.2)
        and launches the PT proxy as a sub-process (fork()/exec()).

     2) The PT Proxy determines the versions of the PT specification
        supported by the parent"TOR_PT_MANAGED_TRANSPORT_VER" (3.2.1)

        2.1) If there are no compatible versions, the PT proxy
             writes a "VERSION-ERROR" message (3.3.1) to stdout and
             terminates.

        2.2) If there is a compatible version, the PT proxy writes
             a "VERSION" message (3.3.1) to stdout.

     3) The PT Proxy parses the rest of the environment values.

        3.1) If the environment values are malformed, or otherwise
             invalid, the PT proxy writes a "ENV-ERROR" message
             (3.3.1) to stdout and terminates.

        3.2) Determining if it is a client side forward proxy or
             a server side reverse proxy can be done via examining
             the "TOR_PT_CLIENT_TRANSPORTS" and
"TOR_PT_SERVER_TRANSPORTS"
             environment variables.

     4) (Client only) If there is an upstream proxy specified via
        "TOR_PT_PROXY" (3.2.2), the PT proxy validates the URI
        provided.

        4.1) If the upstream proxy is unusable, the PT proxy writes
             a "PROXY-ERROR" message (3.3.2) to stdout and
             terminates.

        4.2) If there is a supported and well-formed upstream proxy
             the PT proxy writes a "PROXY DONE" message (3.3.2) to
             stdout.

     5) The PT Proxy initializes the transports and reports the
        status via stdout (3.3.2, 3.3.3)

     6) The PT Proxy forwards and transforms traffic as appropriate.

     7) Upon being signaled to terminate by the parent process (3.4),
        the PT Proxy gracefully shuts down.
```

Pluggable Transport Naming

Pluggable Transport names serve as unique identifiers, and every PT MUST have a unique name.

PT names MUST be valid C identifiers. PT names MUST begin with a letter or underscore, and the remaining characters MUST be ASCII letters, numbers or underscores. No length limit is imposted.

PT names MUST satisfy the regular expression "`[a-zA-Z_][a-zA-Z0-9_]*`".

passwords are isolated on separate circuits if the IsolateSOCKSAuth flag is active; see Proposal 171.)
[Added in Tor 0.4.3.1-alpha.]

The "REND_QUERY" field is provided only for hidden-service-related circuits, and only in versions 0.2.3.11-alpha and later. Clients MUST accept hidden service addresses in formats other than that specified above.
[Added in Tor 0.4.3.1-alpha.]

The "TIME_CREATED" field is provided only in versions 0.2.3.11-alpha and later. TIME_CREATED is the time at which the circuit was created or cannibalized.
[Added in Tor 0.4.3.1-alpha.]

The "REASON" field is provided only for FAILED and CLOSED events, and only if extended events are enabled (see 3.19). Clients MUST accept reasons not listed above.
[Added in Tor 0.4.3.1-alpha.]
Reasons are as given in tor-spec.txt, except for:

- `NOPATH`
  Not enough nodes to make circuit)
- `MEASUREMENT_EXPIRED`
  As "TIMEOUT", except that we had left the circuit open for measurement purposes to see how long it would take to finish.
- `IP_NOW_REDUNDANT`
  Closing a circuit to an introduction point that has become redundant, since some other circuit opened in parallel with it has succeeded.

The "REMOTE_REASON" field is provided only when we receive a DESTROY cell or RELAY_TRUNCATE message, and only if extended events are enabled. It contains the actual reason given by the remote OR for closing the circuit. Clients MUST accept reasons not listed above. Reasons are as listed in tor-spec.txt.
[Added in Tor 0.4.3.1-alpha.]

Stream status changed

The syntax is:

```
      GENERAL         (circuit for AP and/or directory request
streams)
      HS_CLIENT_INTRO (HS client-side introduction-point circuit)
      HS_CLIENT_REND  (HS client-side rendezvous circuit; carries AP
streams)
      HS_SERVICE_INTRO (HS service-side introduction-point circuit)
      HS_SERVICE_REND (HS service-side rendezvous circuit)
      TESTING         (reachability-testing circuit; carries no
traffic)
      CONTROLLER      (circuit built by a controller)
      MEASURE_TIMEOUT (circuit being kept around to see how long it
takes)
      HS_VANGUARDS    (circuit created ahead of time when using
                      HS vanguards, and later repurposed as needed)
      PATH_BIAS_TESTING (circuit used to probe whether our circuits
are
                      being deliberately closed by an attacker)
      CIRCUIT_PADDING (circuit that is being held open to disguise its
                      true close time)
```

The “HS_STATE” field is provided only for hidden-service circuits, and only in versions 0.2.3.11-alpha and later. Clients MUST accept hidden-service circuit states not listed above. Hidden-service circuit states are defined as follows:

```
      HSCI_*      (client-side introduction-point circuit states)
        HSCI_CONNECTING          (connecting to intro point)
        HSCI_INTRO_SENT          (sent INTRODUCE1; waiting for reply
from IP)
        HSCI_DONE                (received reply from IP relay;
closing)

      HSCR_*      (client-side rendezvous-point circuit states)
        HSCR_CONNECTING          (connecting to or waiting for reply
from RP)
        HSCR_ESTABLISHED_IDLE    (established RP; waiting for
introduction)
        HSCR_ESTABLISHED_WAITING (introduction sent to HS; waiting for
rend)
        HSCR_JOINED              (connected to HS)

      HSSI_*      (service-side introduction-point circuit states)
        HSSI_CONNECTING          (connecting to intro point)
        HSSI_ESTABLISHED         (established intro point)

      HSSR_*      (service-side rendezvous-point circuit states)
        HSSR_CONNECTING          (connecting to client's rend point)
        HSSR_JOINED              (connected to client's RP circuit)
```

The “SOCKS_USERNAME” and “SOCKS_PASSWORD” fields indicate the credentials that were used by a SOCKS client to connect to Tor’s SOCKS port and initiate this circuit. (Streams for SOCKS clients connected with different usernames and/or

Pluggable Transport Configuration Environment Variables

All Pluggable Transport proxy instances are configured by their parent process at launch time via a set of well defined environment variables.

The “TOR_PT_” prefix is used for namespacing reasons and does not indicate any relations to Tor, except for the origins of this specification.

Common Environment Variables

When launching either a client or server Pluggable Transport proxy, the following common environment variables MUST be set.

“TOR_PT_MANAGED_TRANSPORT_VER”

Specifies the versions of the Pluggable Transport specification the parent process supports, delimited by commas. All PTs MUST accept any well-formed list, as long as a compatible version is present.

Valid versions MUST consist entirely of non-whitespace, non-comma printable ASCII characters.

The version of the Pluggable Transport specification as of this document is “1”.

Example:

TOR_PT_MANAGED_TRANSPORT_VER=1,1a,2b,this_is_a_valid_ver

“TOR_PT_STATE_LOCATION”

Specifies an absolute path to a directory where the PT is allowed to store state that will be persisted across invocations. The directory is not required to exist when the PT is launched, however PT implementations SHOULD be able to create it as required.

PTs MUST only store files in the path provided, and MUST NOT create or modify files elsewhere on the system.

Example:

TOR_PT_STATE_LOCATION=/var/lib/tor/pt_state/

“TOR_PT_EXIT_ON_STDIN_CLOSE”

Specifies that the parent process will close the PT proxy’s standard input (stdin) stream to indicate that the PT proxy should gracefully exit.

PTs MUST NOT treat a closed stdin as a signal to terminate unless this environment variable is set to “1”.

PTs SHOULD treat stdin being closed as a signal to gracefully terminate if this environment variable is set to “1”.

Example:

TOR_PT_EXIT_ON_STDIN_CLOSE=1

“TOR_PT_OUTBOUND_BIND_ADDRESS_V4”

Specifies an IPv4 IP address that the PT proxy SHOULD use as source address for outgoing IPv4 IP packets. This feature allows people with multiple network interfaces to specify explicitly which interface they prefer the PT proxy to use.

If this value is unset or empty, the PT proxy MUST use the default source address for outgoing connections.

This setting MUST be ignored for connections to loopback addresses (127.0.0.0/8).

Example:

TOR_PT_OUTBOUND_BIND_ADDRESS_V4=203.0.113.4

“TOR_PT_OUTBOUND_BIND_ADDRESS_V6”

Specifies an IPv6 IP address that the PT proxy SHOULD use as source address for outgoing IPv6 IP packets. This feature allows people with multiple network interfaces to specify explicitly which interface they prefer the PT proxy to use.

If this value is unset or empty, the PT proxy MUST use the default source address for outgoing connections.

This setting MUST be ignored for connections to the loopback address ([::1]).

IPv6 addresses MUST always be wrapped in square brackets.

Example::

```
      EscapedUsername = QuotedString
      EscapedPassword = QuotedString

      HSAddress = 16*Base32Character / 56*Base32Character
      Base32Character = ALPHA / "2" / "3" / "4" / "5" / "6" / "7"

      TimeCreated = ISOTime2Frac
      Seconds = 1*DIGIT
      Microseconds = 1*DIGIT

      Reason = "NONE" / "TORPROTOCOL" / "INTERNAL" / "REQUESTED" /
               "HIBERNATING" / "RESOURCELIMIT" / "CONNECTFAILED" /
               "OR_IDENTITY" / "OR_CONN_CLOSED" / "TIMEOUT" /
               "FINISHED" / "DESTROYED" / "NOPATH" / "NOSUCHSERVICE" /
               "MEASUREMENT_EXPIRED"
```

The path is provided only when the circuit has been extended at least one hop.

The “BUILD_FLAGS” field is provided only in versions 0.2.3.11-alpha and later. Clients MUST accept build flags not listed above. Build flags are defined as follows:

```
      ONEHOP_TUNNEL   (one-hop circuit, used for tunneled directory
conns)
      IS_INTERNAL     (internal circuit, not to be used for exiting
streams)
      NEED_CAPACITY   (this circuit must use only high-capacity nodes)
      NEED_UPTIME     (this circuit must use only high-uptime nodes)
```

The “PURPOSE” field is provided only in versions 0.2.1.6-alpha and later, and only if extended events are enabled (see 3.19). Clients MUST accept purposes not listed above. Purposes are defined as follows:

```
     "650" SP "CIRC" SP CircuitID SP CircStatus [SP Path]
          [SP "BUILD_FLAGS=" BuildFlags] [SP "PURPOSE=" Purpose]
          [SP "HS_STATE=" HSState] [SP "REND_QUERY=" HSAddress]
          [SP "TIME_CREATED=" TimeCreated]
          [SP "REASON=" Reason [SP "REMOTE_REASON=" Reason]]
          [SP "SOCKS_USERNAME=" EscapedUsername]
          [SP "SOCKS_PASSWORD=" EscapedPassword]
          [SP "HS_POW=" HSPoW ]
          [SP "CONFLUX_ID=" ConfluxID ]
          [SP "CONFLUX_RTT=" ConfluxRTT ]
          CRLF

     CircStatus =
               "LAUNCHED" / ; circuit ID assigned to new circuit
               "BUILT"    / ; all hops finished, can now accept
streams
               "GUARD_WAIT" / ; all hops finished, waiting to see if a
                              ;  circuit with a better guard will be
usable.
               "EXTENDED" / ; one more hop has been completed
               "FAILED"   / ; circuit closed (was not built)
               "CLOSED"     ; circuit closed (was built)

     Path = LongName *("," LongName)
        ; In Tor versions 0.1.2.2-alpha through 0.2.2.1-alpha with
feature
        ; VERBOSE_NAMES turned off and before version 0.1.2.2-alpha,
Path
        ; is as follows:
        ; Path = ServerID *("," ServerID)

     BuildFlags = BuildFlag *("," BuildFlag)
     BuildFlag = "ONEHOP_TUNNEL" / "IS_INTERNAL" /
                 "NEED_CAPACITY" / "NEED_UPTIME"

     Purpose = "GENERAL" / "HS_CLIENT_INTRO" / "HS_CLIENT_REND" /
               "HS_SERVICE_INTRO" / "HS_SERVICE_REND" / "TESTING" /
               "CONTROLLER" / "MEASURE_TIMEOUT" /
               "HS_VANGUARDS" / "PATH_BIAS_TESTING" /
               "CIRCUIT_PADDING"

     HSState = "HSCI_CONNECTING" / "HSCI_INTRO_SENT" / "HSCI_DONE" /
               "HSCR_CONNECTING" / "HSCR_ESTABLISHED_IDLE" /
               "HSCR_ESTABLISHED_WAITING" / "HSCR_JOINED" /
               "HSSI_CONNECTING" / "HSSI_ESTABLISHED" /
               "HSSR_CONNECTING" / "HSSR_JOINED"

     HSPoWType = "v1"
     HSPoWEffort = 1*DIGIT
     HSPoW = HSPoWType "," HSPoWEffort

     ConfluxID = 32*HEXDIG
     ConfluxRTT = UInt
```

TOR_PT_OUTBOUND_BIND_ADDRESS_V6=[2001:db8::4]

Pluggable Transport Client Environment Variables

Client-side Pluggable Transport forward proxies are configured via the following environment variables.

“TOR_PT_CLIENT_TRANSPORTS”

Specifies the PT protocols the client proxy should initialize, as a comma separated list of PT names.

PTs SHOULD ignore PT names that it does not recognize.

Parent processes MUST set this environment variable when launching a client-side PT proxy instance.

Example:

TOR_PT_CLIENT_TRANSPORTS=obfs2,obfs3,obfs4

“TOR_PT_PROXY”

Specifies an upstream proxy that the PT MUST use when making outgoing network connections. It is a URI [RFC3986] of the format:

`<proxy_type>://[<user_name>[:<password>][@]<ip>:<port>.`

The “TOR_PT_PROXY” environment variable is OPTIONAL and MUST be omitted if there is no need to connect via an upstream proxy.

Examples:

```
TOR_PT_PROXY=socks5://tor:test1234@198.51.100.1:8000
TOR_PT_PROXY=socks4a://198.51.100.2:8001
TOR_PT_PROXY=http://198.51.100.3:443
```

Pluggable Transport Server Environment Variables

Server-side Pluggable Transport reverse proxies are configured via the following environment variables.

“TOR_PT_SERVER_TRANSPORTS”

Specifies the PT protocols the server proxy should initialize, as a comma separated list of PT names.

PTs SHOULD ignore PT names that it does not recognize.

Parent processes MUST set this environment variable when launching a server-side PT reverse proxy instance.

Example:

TOR_PT_SERVER_TRANSPORTS=obfs3,scramblesuit

“TOR_PT_SERVER_TRANSPORT_OPTIONS”

Specifies per-PT protocol configuration directives, as a semicolon-separated list of `<key>:<value>` pairs, where `<key>` is a PT name and `<value>` is a k=v string value with options that are to be passed to the transport.

Colons, semicolons, and backslashes MUST be escaped with a backslash.

If there are no arguments that need to be passed to any of PT transport protocols, “TOR_PT_SERVER_TRANSPORT_OPTIONS” MAY be omitted.

Example:

TOR_PT_SERVER_TRANSPORT_OPTIONS=scramblesuit:key=banana;automata:rule=110;automata:depth=3

```
Will pass to 'scramblesuit' the parameter 'key=banana' and to
'automata' the arguments 'rule=110' and 'depth=3'.
```

`"TOR_PT_SERVER_BINDADDR"`

A comma separated list of `<key>-<value>` pairs, where `<key>` is a PT name and `<value>` is the `<address>:<port>` on which it should listen for incoming client connections.

The keys holding transport names MUST be in the same order as they appear in “TOR_PT_SERVER_TRANSPORTS”.

The `<address>` MAY be a locally scoped address as long as port forwarding is done externally.

The `<address>:<port>` combination MUST be an IP address supported by `bind()`, and MUST NOT be a host name.

Clients MUST tolerate more arguments in an asynchronous reply than expected, and MUST tolerate more lines in an asynchronous reply than expected. For instance, a client that expects a CIRC message like:

```
650 CIRC 1000 EXTENDED moria1,moria2
```

must tolerate:

```
650-CIRC 1000 EXTENDED moria1,moria2 0xBEEF
650-EXTRAMAGIC=99
650 ANONYMITY=high
```

If clients receives extended events (selected by USEFEATUERE EXTENDED_EVENTS in Tor 0.1.2.2-alpha..Tor-0.2.1.x, and always-on in Tor 0.2.2.x and later), then each event line as specified below may be followed by additional arguments and additional lines.

Additional lines will be of the form:

```
"650" ("-" / " ") KEYWORD ["=" ARGUMENTS] CRLF
```

Additional arguments will be of the form

```
SP KEYWORD \["=" ( QuotedString / * NonSpDquote ) \]
```

Clients MUST tolerate events with arguments and keywords they do not recognize, and SHOULD process those events as if any unrecognized arguments and keywords were not present.

Clients SHOULD NOT depend on the order of keyword=value arguments, and SHOULD NOT depend on there being no new keyword=value arguments appearing between existing keyword=value arguments, though as of this writing (Jun 2011) some do. Thus, extensions to this protocol should add new keywords only after the existing keywords, until all controllers have been fixed.
At some point this “SHOULD NOT” might become a “MUST NOT”.

Circuit status changed

The syntax is:

- `514` **Authentication required**
- `515` **Bad authentication**
- `550` **Unspecified Tor error**
- `551` **Internal error**
  Something went wrong inside Tor, so that the client's request couldn't be fulfilled.
- `552` **Unrecognized entity**
  A configuration key, a stream ID, circuit ID, event, mentioned in the command did not actually exist.
- `553` **Invalid configuration value**
  The client tried to set a configuration option to an incorrect, ill-formed, or impossible value.
- `554` **Invalid descriptor**
- `555` **Unmanaged entity**
- `650` **Asynchronous event notification**

Unless specified to have specific contents, the human-readable messages in error replies should not be relied upon to match those in this document.

Asynchronous events

These replies can be sent after a corresponding SETEVENTS command has been received. They will not be interleaved with other Reply elements, but they can appear between a command and its corresponding reply. For example, this sequence is possible:

```
C: SETEVENTS CIRC
S: 250 OK
C: GETCONF SOCKSPORT ORPORT
S: 650 CIRC 1000 EXTENDED moria1,moria2
S: 250-SOCKSPORT=9050
S: 250 ORPORT=0
```

But this sequence is disallowed:

```
C: SETEVENTS CIRC
S: 250 OK
C: GETCONF SOCKSPORT ORPORT
S: 250-SOCKSPORT=9050
S: 650 CIRC 1000 EXTENDED moria1,moria2
S: 250 ORPORT=0
```

Applications MUST NOT set more than one `<address>:<port>` pair per PT name.

If there is no specific `<address>:<port>` combination to be configured for any transports, "TOR_PT_SERVER_BINDADDR" MAY be omitted.

Example:

TOR_PT_SERVER_BINDADDR=obfs3-198.51.100.1:1984,scramblesuit-127.0.0.1:4891

"TOR_PT_ORPORT"

Specifies the destination that the PT reverse proxy should forward traffic to after transforming it as appropriate, as an `<address>:<port>`.

Connections to the destination specified via "TOR_PT_ORPORT" MUST only contain application payload. If the parent process requires the actual source IP address of client connections (or other metadata), it should set "TOR_PT_EXTENDED_SERVER_PORT" instead.

Example:

TOR_PT_ORPORT=127.0.0.1:9001

"TOR_PT_EXTENDED_SERVER_PORT"

Specifies the destination that the PT reverse proxy should forward traffic to, via the Extended ORPort protocol [EXTORPORT] as an `<address>:<port>`.

The Extended ORPort protocol allows the PT reverse proxy to communicate per-connection metadata such as the PT name and client IP address/port to the parent process.

If the parent process does not support the ExtORPort protocol, it MUST set "TOR_PT_EXTENDED_SERVER_PORT" to an empty string.

Example:

TOR_PT_EXTENDED_SERVER_PORT=127.0.0.1:4200

"TOR_PT_AUTH_COOKIE_FILE"

Specifies an absolute filesystem path to the Extended ORPort authentication cookie, required to communicate with the Extended ORPort specified via "TOR_PT_EXTENDED_SERVER_PORT".

If the parent process is not using the ExtORPort protocol for incoming traffic, “TOR_PT_AUTH_COOKIE_FILE” MUST be omitted.

Example:

TOR_PT_AUTH_COOKIE_FILE=/var/lib/tor/extended_orport_auth_cookie

Replies

Reply codes follow the same 3-character format as used by SMTP, with the first character defining a status, the second character defining a subsystem, and the third designating fine-grained information.

The TC protocol currently uses the following first characters:

- `2yz` **Positive Completion Reply**
  The command was successful; a new request can be started.
- `4yz` **Temporary Negative Completion reply**
  The command was unsuccessful but might be reattempted later.
- `5yz` **Permanent Negative Completion Reply**
  The command was unsuccessful; the client should not try exactly that sequence of commands again.
- `6yz` **Asynchronous Reply**
  Sent out-of-order in response to an earlier SETEVENTS command.

The following second characters are used:

- `x0z` **Syntax** Sent in response to ill-formed or nonsensical commands.
- `x1z` **Protocol**
  Refers to operations of the Tor Control protocol.
- `x5z` **Tor**
  Refers to actual operations of Tor system.

The following codes are defined:

- `250` **OK**
- `251` **Operation was unnecessary**
  Tor has declined to perform the operation, but no harm was done.
- `451` **Resource exhausted**
- `500` **Syntax error: protocol**
- `510` **Unrecognized command**
- `511` **Unimplemented command**
- `512` **Syntax error in command argument**
- `513` **Unrecognized command argument**

[ONION_CLIENT_AUTH_VIEW was added in Tor 0.4.3.1-alpha]

DROPOWNERSHIP

The syntax is:

```
"DROPOWNERSHIP" CRLF
```

This command instructs Tor to relinquish ownership of its control connection. As such tor will not shut down when this control connection is closed.

This method is idempotent. If the control connection does not already have ownership this method returns successfully, and does nothing.

The controller can call TAKEOWNERSHIP again to re-establish ownership.

[DROPOWNERSHIP was added in Tor 0.4.0.0-alpha]

DROPTIMEOUTS

The syntax is:

```
"DROPTIMEOUTS" CRLF
```

Tells the server to drop all circuit build times. Do not invoke this command lightly; it can increase vulnerability to tracking attacks over time.

Tor replies with `250 OK` on success. Tor also emits the BUILDTIMEOUT_SET RESET event right after this `250 OK`.

[DROPTIMEOUTS was added in Tor 0.4.5.0-alpha.]

Pluggable Transport To Parent Process Communication

All Pluggable Transport Proxies communicate to the parent process via writing NL-terminated lines to stdout. The line metaformat is:

```
<Line> ::= <Keyword> <OptArgs> <NL>
<Keyword> ::= <KeywordChar> | <Keyword> <KeywordChar>
<KeywordChar> ::= <any US-ASCII alphanumeric, dash, and underscore>
<OptArgs> ::= <Args>*
<Args> ::= <SP> <ArgChar> | <Args> <ArgChar>
<ArgChar> ::= <any US-ASCII character but NUL or NL>
<SP> ::= <US-ASCII whitespace symbol (32)>
<NL> ::= <US-ASCII newline (line feed) character (10)>
```

The parent process MUST ignore lines received from PT proxies with unknown keywords.

Common Messages

When a PT proxy first starts up, it must determine which version of the Pluggable Transports Specification to use to configure itself.

It does this via the “TOR_PT_MANAGED_TRANSPORT_VER” (3.2.1) environment variable which contains all of the versions supported by the application.

Upon determining the version to use, or lack thereof, the PT proxy responds with one of two messages.

```
VERSION-ERROR <ErrorMessage>
```

The “VERSION-ERROR” message is used to signal that there was no compatible Pluggable Transport Specification version present in the “TOR_PT_MANAGED_TRANSPORT_VER” list.

The `<ErrorMessage>` SHOULD be set to “no-version” for historical reasons but MAY be set to a useful error message instead.

PT proxies MUST terminate after outputting a “VERSION-ERROR” message.

Example:

```
VERSION-ERROR no-version

VERSION <ProtocolVersion>
```

The “VERSION” message is used to signal the Pluggable Transport Specification version (as in “TOR_PT_MANAGED_TRANSPORT_VER”) that the PT proxy will use to configure its transports and communicate with the parent process.

The version for the environment values and reply messages specified by this document is “1”.

PT proxies MUST either report an error and terminate, or output a “VERSION” message before moving on to client/server proxy initialization and configuration.

Example:

VERSION 1

After version negotiation has been completed the PT proxy must then validate that all of the required environment variables are provided, and that all of the configuration values supplied are well formed.

At any point, if there is an error encountered related to configuration supplied via the environment variables, it MAY respond with an error message and terminate.

```
ENV-ERROR <ErrorMessage>
```

The “ENV-ERROR” message is used to signal the PT proxy’s failure to parse the configuration environment variables (3.2).

The `<ErrorMessage>` SHOULD consist of a useful error message that can be used to diagnose and correct the root cause of the failure.

PT proxies MUST terminate after outputting a “ENV-ERROR” message.

Example:

ENV-ERROR No TOR_PT_AUTH_COOKIE_FILE when
TOR_PT_EXTENDED_SERVER_PORT set

Pluggable Transport Client Messages

After negotiating the Pluggable Transport Specification version, PT client proxies MUST first validate “TOR_PT_PROXY” (3.2.2) if it is set, before initializing any transports.

Tells the connected Tor to remove the client-side v3 client auth credentials for the onion service with “HSAddress”.

On success `250 OK` is returned. Otherwise, the following error codes exist:

- `512` - Syntax error in “HSAddress”.
- `251` - Client credentials for “HSAddress” did not exist.

[ONION_CLIENT_AUTH_REMOVE was added in Tor 0.4.3.1-alpha]

ONION_CLIENT_AUTH_VIEW

The syntax is:

```
    "ONION_CLIENT_AUTH_VIEW" [SP HSAddress] CRLF
```

Tells the connected Tor to list all the stored client-side v3 client auth credentials for “HSAddress”. If no “HSAddress” is provided, list all the stored client-side v3 client auth credentials.

The server reply format is:

```
    "250-ONION_CLIENT_AUTH_VIEW" [SP HSAddress] CRLF
    *("250-CLIENT" SP HSAddress SP KeyType ":" PrivateKeyBlob
                  [SP "ClientName=" Nickname]
                  [SP "Flags=" FLAGS] CRLF)
    "250 OK" CRLF

    HSAddress = The onion address under which this credential is
stored
    KeyType = "x25519" is the only one supported right now
    PrivateKeyBlob = base64 encoding of x25519 key
```

“Nickname” is an optional nickname for this client, which can be set either through the ONION_CLIENT_AUTH_ADD command, or it’s the filename of this client if the credentials are stored in the filesystem.

FLAGS is a comma-separated field of flags for this client, the currently supported flags are:

“Permanent” - This client’s credentials are stored in the filesystem.

On success `250 OK` is returned. Otherwise, the following error codes exist:

- `512` - Syntax error in “HSAddress”.

Tells the connected Tor to add client-side v3 client auth credentials for the onion service with “HSAddress”. The “PrivateKeyBlob” is the x25519 private key that should be used for this client, and “Nickname” is an optional nickname for the client.

FLAGS is a comma-separated tuple of flags for this new client. For now, the currently supported flags are:

```
    "Permanent" - This client's credentials should be stored in the
filesystem.
                  If this is not set, the client's credentials are
ephemeral
                  and stored in memory.
```

If client auth credentials already existed for this service, replace them with the new ones.

If Tor has cached onion service descriptors that it has been unable to decrypt in the past (due to lack of client auth credentials), attempt to decrypt those descriptors as soon as this command succeeds.

On success, `250 OK` is returned. Otherwise, the following error codes exist:

- `251` - Client auth credentials for this onion service already existed and replaced.
- `252` - Added client auth credentials and successfully decrypted a cached descriptor.
- `451` - We reached authorized client capacity
- `512` - Syntax error in “HSAddress”, or “PrivateKeyBlob” or “Nickname”
- `551` - Client with with this “Nickname” already exists
- `552` - Unrecognized KeyType

[ONION_CLIENT_AUTH_ADD was added in Tor 0.4.3.1-alpha]

ONION_CLIENT_AUTH_REMOVE

The syntax is:

```
    "ONION_CLIENT_AUTH_REMOVE" SP HSAddress CRLF
```

KeyType = “x25519” is the only one supported right now

Assuming that an upstream proxy is provided, PT client proxies MUST respond with a message indicating that the proxy is valid, supported, and will be used OR a failure message.

PROXY DONE

The “PROXY DONE” message is used to signal the PT proxy’s acceptance of the upstream proxy specified by “TOR_PT_PROXY”.

```
PROXY-ERROR <ErrorMessage>
```

The “PROXY-ERROR” message is used to signal that the upstream proxy is malformed/unsupported or otherwise unusable.

PT proxies MUST terminate immediately after outputting a “PROXY-ERROR” message.

Example:

PROXY-ERROR SOCKS 4 upstream proxies unsupported.

After the upstream proxy (if any) is configured, PT clients then iterate over the requested transports in “TOR_PT_CLIENT_TRANSPORTS” and initialize the listeners.

For each transport initialized, the PT proxy reports the listener status back to the parent via messages to stdout.

```
CMETHOD <transport> <'socks4','socks5'> <address:port>
```

The “CMETHOD” message is used to signal that a requested PT transport has been launched, the protocol which the parent should use to make outgoing connections, and the IP address and port that the PT transport’s forward proxy is listening on.

Example:

```
CMETHOD trebuchet socks5 127.0.0.1:19999
```

```
CMETHOD-ERROR <transport> <ErrorMessage>
```

The “CMETHOD-ERROR” message is used to signal that requested PT transport was unable to be launched.

Example:

CMETHOD-ERROR trebuchet no rocks available

Once all PT transports have been initialized (or have failed), the PT proxy MUST send a final message indicating that it has finished initializing.

CMETHODS DONE

The “CMETHODS DONE” message signals that the PT proxy has finished initializing all of the transports that it is capable of handling.

Upon sending the “CMETHODS DONE” message, the PT proxy initialization is complete.

Notes:

- Unknown transports in "TOR_PT_CLIENT_TRANSPORTS" are ignored entirely, and MUST NOT result in a "CMETHOD-ERROR" message. Thus it is entirely possible for a given PT proxy to immediately output "CMETHODS DONE".
- Parent processes MUST handle "CMETHOD"/"CMETHOD-ERROR" messages in any order, regardless of ordering in "TOR_PT_CLIENT_TRANSPORTS".

Pluggable Transport Server Messages

PT server reverse proxies iterate over the requested transports in “TOR_PT_CLIENT_TRANSPORTS” and initialize the listeners.

For each transport initialized, the PT proxy reports the listener status back to the parent via messages to stdout.

```
SMETHOD <transport> <address:port> [options]
```

The “SMETHOD” message is used to signal that a requested PT transport has been launched, the protocol which will be used to handle incoming connections, and the IP address and port that clients should use to reach the reverse-proxy.

If there is a specific address:port provided for a given PT transport via “TOR_PT_SERVER_BINDADDR”, the transport MUST be initialized using that as the server address.

The OPTIONAL ‘options’ field is used to pass additional per-transport information back to the parent process.

The currently recognized ‘options’ are:

```
"+HSPOST" *[SP "SERVER=" Server] [SP "HSADDRESS=" HSAddress]
          CRLF Descriptor CRLF "." CRLF

Server = LongName
HSAddress = 56*Base32Character
Descriptor =  The text of the descriptor formatted as specified
              in rend-spec.txt section 1.3.
```

The “HSAddress” key is optional and only applies for v3 descriptors. A `513` error is returned if used with v2.

This command launches a hidden service descriptor upload to the specified HSDirs. If one or more Server arguments are provided, an upload is triggered on each of them in parallel. If no Server options are provided, it behaves like a normal HS descriptor upload and will upload to the set of responsible HS directories.

If any value is unrecognized, a `552` error is returned and the command is stopped. If there is an error in parsing the descriptor, the server must send a `554 Invalid descriptor` reply.

On success, Tor replies `250 OK` then Tor MUST eventually follow this with an HS_DESC event with the result for each upload location.

Examples are:

```
C: +HSPOST SERVER=9695DFC35FFEB861329B9F1AB04C46397020CE31
   [DESCRIPTOR]
   .
S: 250 OK
```

```
[HSPOST was added in Tor 0.2.7.1-alpha]
```

ONION_CLIENT_AUTH_ADD

The syntax is:

```
"ONION_CLIENT_AUTH_ADD" SP HSAddress
                        SP KeyType ":" PrivateKeyBlob
                        [SP "ClientName=" Nickname]
                        [SP "Flags=" TYPE] CRLF

HSAddress = 56*Base32Character
KeyType = "x25519" is the only one supported right now
PrivateKeyBlob = base64 encoding of x25519 key
```

[ClientAuth was added in Tor 0.2.9.1-alpha. It is v2 only.]

[NonAnonymous was added in Tor 0.2.9.3-alpha.]

[HS v3 support added 0.3.3.1-alpha]

[ClientV3Auth support added 0.4.6.1-alpha]

[PoWDefensesEnabled, PoWQueueRate and PoWQueueBurst support added 0.4.9.2-alpha]

DEL_ONION

The syntax is:

```
    "DEL_ONION" SP ServiceID CRLF

    ServiceID = The Onion Service address without the trailing
".onion"
                suffix
```

Tells the server to remove an Onion (“Hidden”) Service, that was previously created via an “ADD_ONION” command. It is only possible to remove Onion Services that were created on the same control connection as the “DEL_ONION” command, and those that belong to no control connection in particular (The “Detach” flag was specified at creation).

If the ServiceID is invalid, or is neither owned by the current control connection nor a detached Onion Service, the server will return a `552`.

It is the Onion Service server application’s responsibility to close existing client connections if desired after the Onion Service has been removed via “DEL_ONION”.

Tor replies with `250 OK` on success, or a `512` if there are an invalid number of arguments, or a `552` if it doesn’t recognize the ServiceID.

[DEL_ONION was added in Tor 0.2.7.1-alpha.] [HS v3 support added 0.3.3.1-alpha]

HSPOST

The syntax is:

```
ARGS:[<Key>=<Value>,]+[<Key>=<Value>]

          The "ARGS" option is used to pass additional key/value
          formatted information that clients will require to use
          the reverse proxy.

          Equal signs and commas MUST be escaped with a backslash.

          Tor: The ARGS are included in the transport line of the
          Bridge's extra-info document.

      Examples:

        SMETHOD trebuchet 198.51.100.1:19999
        SMETHOD rot_by_N 198.51.100.1:2323 ARGS:N=13

    SMETHOD-ERROR <transport> <ErrorMessage>
```

The “SMETHOD-ERROR” message is used to signal that requested PT transport reverse proxy was unable to be launched.

Example:

SMETHOD-ERROR trebuchet no cows available

Once all PT transports have been initialized (or have failed), the PT proxy MUST send a final message indicating that it has finished initializing.

SMETHODS DONE

The “SMETHODS DONE” message signals that the PT proxy has finished initializing all of the transports that it is capable of handling.

Upon sending the “SMETHODS DONE” message, the PT proxy initialization is complete.

Pluggable Transport Log Message

This message is for a client or server PT to be able to signal back to the parent process via stdout or stderr any log messages.

A log message can be any kind of messages (human readable) that the PT sends back so the parent process can gather information about what is going on in the child process. It is not intended for the parent process to parse and act accordingly but rather a message used for plain logging.

For example, the tor daemon logs those messages at the Severity level and sends them onto the control port using the PT_LOG (see control-spec.txt) event so any third party can pick them up for debugging.

The format of the message:

```
LOG SEVERITY=Severity MESSAGE=Message
```

The SEVERITY value indicate at which logging level the message applies. The accepted values for `<Severity>` are: error, warning, notice, info, debug

The MESSAGE value is a human readable string formatted by the PT. The `<Message>` contains the log message which can be a String or CString (see section 2 in control-spec.txt).

Example:

```
LOG SEVERITY=debug MESSAGE="Connected to bridge A"
```

Pluggable Transport Status Message

This message is for a client or server PT to be able to signal back to the parent process via stdout or stderr any status messages.

The format of the message:

```
STATUS [TRANSPORT=Transport] TYPE=Type [<K_1>=<V_1> [<K_2>=<V_2> ...]]
```

The TYPE value indicates the status message type. Each Type has its own set of associated `<K_n>=<V_n>` values. Type can be a String or CString.

Following the TYPE specification are zero or more key=value pairs separated by spaces. Each TYPE has its own set of known keys. Each `<V_n>` can be a String or CString. To avoid confusion with the TYPE=Type specification, no `<K_n>` may be “TYPE”. TYPE and the different K_n values may appear in any order.

The parent process should ignore STATUS messages that have a TYPE it does not understand. The parent process should ignore STATUS messages that do not have a TYPE set, or that have two or more TYPE=Type specifications.

> Compatibility note:
>
> Versions of Tor and Arti released before August of 2024 treated the TRANSPORT key as required, and rejected any status message that did not

```
     C: ADD_ONION NEW:BEST Flags=DiscardPK Port=80
     S: 250-
ServiceID=exampleoniont2pqglbny66wpovyvao3ylc23eileodtevc4b75ikpad
     S: 250 OK

     C: ADD_ONION RSA1024:[Blob Redacted] Port=80,192.168.1.1:8080
     S: 250-ServiceID=sampleonion12456
     S: 250 OK

     C: ADD_ONION NEW:BEST Port=22 Port=80,8080
     S: 250-
ServiceID=sampleonion4t2pqglbny66wpovyvao3ylc23eileodtevc4b75ikpad
     S: 250-PrivateKey=ED25519-V3:[Blob Redacted]
     S: 250 OK

     C: ADD_ONION NEW:RSA1024 Flags=DiscardPK,BasicAuth Port=22
        ClientAuth=alice:[Blob Redacted] ClientAuth=bob
     S: 250-ServiceID=testonion1234567
     S: 250-ClientAuth=bob:[Blob Redacted]
     S: 250 OK

     C: ADD_ONION NEW:ED25519-V3 ClientAuthV3=[Blob Redacted] Port=22
     S: 250-
ServiceID=n35etu3yjxrqjpntmfziom5sjwspoydchmelc4xleoy4jk2u4lziz2yd
     S: 250-ClientAuthV3=[Blob Redacted]
     S: 250 OK
```

Examples with Tor in anonymous onion service mode:

```
     C: ADD_ONION NEW:BEST Flags=DiscardPK Port=22
     S: 250-
ServiceID=exampleoniont2pqglbny66wpovyvao3ylc23eileodtevc4b75ikpad
     S: 250 OK

     C: ADD_ONION NEW:BEST Flags=DiscardPK,NonAnonymous Port=22
     S: 512 Tor is in anonymous hidden service mode
```

Examples with Tor in non-anonymous onion service mode:

```
     C: ADD_ONION NEW:BEST Flags=DiscardPK Port=22
     S: 512 Tor is in non-anonymous hidden service mode

     C: ADD_ONION NEW:BEST Flags=DiscardPK,NonAnonymous Port=22
     S: 250-
ServiceID=exampleoniont2pqglbny66wpovyvao3ylc23eileodtevc4b75ikpad
     S: 250 OK
```

[ADD_ONION was added in Tor 0.2.7.1-alpha.]

[MaxStreams and MaxStreamsCloseCircuit were added in Tor 0.2.7.2-alpha]

If client authorization is enabled using the “BasicAuth” flag (which is v2 only), the service will not be accessible to clients without valid authorization data (configured with the “HidServAuth” option). The list of authorized clients is specified with one or more “ClientAuth” parameters. If “ClientBlob” is not specified for a client, a new credential will be randomly generated and returned.

Tor instances can either be in anonymous hidden service mode, or non-anonymous single onion service mode. All hidden services on the same tor instance have the same anonymity. To guard against unexpected loss of anonymity, Tor checks that the ADD_ONION “NonAnonymous” flag matches the current hidden service anonymity mode. The hidden service anonymity mode is configured using the Tor options HiddenServiceSingleHopMode and HiddenServiceNonAnonymousMode. If both these options are 1, the “NonAnonymous” flag must be provided to ADD_ONION. If both these options are 0 (the Tor default), the flag must NOT be provided.

Once created the new Onion Service will remain active until either the Onion Service is removed via “DEL_ONION”, the server terminates, or the control connection that originated the “ADD_ONION” command is closed. It is possible to override disabling the Onion Service on control connection close by specifying the “Detach” flag.

It is the Onion Service server application’s responsibility to close existing client connections if desired after the Onion Service is removed.

(The KeyBlob format is left intentionally opaque, however for “RSA1024” keys it is currently the Base64 encoded DER representation of a PKCS#1 RSAPrivateKey, with all newlines removed. For a “ED25519-V3” key is the Base64 encoding of the concatenation of the 32-byte ed25519 secret scalar in little-endian and the 32-byte ed25519 PRF secret.)

[Note: The ED25519-V3 format is not the same as, e.g., SUPERCOP ed25519/ref, which stores the concatenation of the 32-byte ed25519 hash seed concatenated with the 32-byte public key, and which derives the secret scalar and PRF secret by expanding the hash seed with SHA-512. Our key blinding scheme is incompatible with storing private keys as seeds, so we store the secret scalar alongside the PRF secret, and just pay the cost of recomputing the public key when importing an ED25519-V3 key.]

Examples:

contain it. If you need to write a pluggable transport to work with one of those versions, you need to provide a value in the TRANSPORT field.

Here are the recognized TYPEs, and the keys that are understood for them.

TYPE=version

Required keys: IMPLEMENTATION, VERSION

The VERSION type reports the name of the PT implementation and its version number. One of the uses of this message is to enable contacting bridge operators that are running an out-of-date pluggable transport implementation.

IMPLEMENTATION is the name of the software package implementing the pluggable transport (because different implementations may have different version numbering schemes).

VERSION is the version number of the software package implementing the pluggable transport.

A single STATUS message of the VERSION type may be sent any time before “CMETHODS DONE” or “SMETHODS DONE”. If sent after that, the parent process should ignore the message. If a VERSION message is sent more than once, only the first one counts.

Examples:

```
STATUS TYPE=version VERSION=0.0.13 IMPLEMENTATION=obfs4proxy
STATUS IMPLEMENTATION=snowflake-client VERSION=2.1.0 TYPE=version
```

Pluggable Transport Shutdown

The recommended way for Pluggable Transport using applications and Pluggable Transports to handle graceful shutdown is as follows.

```
- (Parent) Set "TOR_PT_EXIT_ON_STDIN_CLOSE" (3.2.1) when
  launching the PT proxy, to indicate that stdin will be used
  for graceful shutdown notification.

- (Parent) When the time comes to terminate the PT proxy:

  1. Close the PT proxy's stdin.
  2. Wait for a "reasonable" amount of time for the PT to exit.
  3. Attempt to use OS specific mechanisms to cause graceful
     PT shutdown (eg: 'SIGTERM')
  4. Use OS specific mechanisms to force terminate the PT
     (eg: 'SIGKILL', 'ProccessTerminate()').

- PT proxies SHOULD monitor stdin, and exit gracefully when
  it is closed, if the parent supports that behavior.

- PT proxies SHOULD handle OS specific mechanisms to gracefully
  terminate (eg: Install a signal handler on 'SIGTERM' that
  causes cleanup and a graceful shutdown if able).

- PT proxies SHOULD attempt to detect when the parent has
  terminated (eg: via detecting that its parent process ID has
  changed on U*IX systems), and gracefully terminate.
```

```
    VirtPort = The virtual TCP Port for the Onion Service (As in the
               HiddenServicePort "VIRTPORT" argument).

    Target = The (optional) target for the given VirtPort (As in the
             optional HiddenServicePort "TARGET" argument).

    ClientName = An identifier 1 to 16 characters long, using only
                 characters in A-Za-z0-9+-_ (no spaces) (v2 only).

    ClientBlob = Authorization data for the client, in an opaque
format
                 specific to the authorization method (v2 only).

    V3Key = The client's base32-encoded x25519 public key, using only
the key
            part of rend-spec-v3.txt section G.1.2 (v3 only).

    PowEnabled = Either 0 to disable or 1 to enable Proof-of-Work
defenses.
                 Default if not present is 0, disabled.

    PowQRate = The sustained rate of rendezvous requests to dispatch
per
               second from the priority queue. Default value 250.

    PowQBurst = The maximum burst size for rendezvous requests handled
from
                the priority queue at once. Default is 2500.
```

The server reply format is:

```
    "250-ServiceID=" ServiceID CRLF
    ["250-PrivateKey=" KeyType ":" KeyBlob CRLF]
    *("250-ClientAuth=" ClientName ":" ClientBlob CRLF)
    "250 OK" CRLF

    ServiceID = The Onion Service address without the trailing
".onion"
                suffix
```

Tells the server to create a new Onion (“Hidden”) Service, with the specified private key and algorithm. If a KeyType of “NEW” is selected, the server will generate a new keypair using the selected algorithm. The “Port” argument’s VirtPort and Target values have identical semantics to the corresponding HiddenServicePort configuration values.

The server response will only include a private key if the server was requested to generate a new keypair, and also the “DiscardPK” flag was not specified. (Note that if “DiscardPK” flag is specified, there is no way to recreate the generated keypair and the corresponding Onion Service at a later date).

```
    "ADD_ONION" SP KeyType ":" KeyBlob
            [SP "Flags=" Flag *("," Flag)]
            [SP "MaxStreams=" NumStreams]
            [SP "PoWDefensesEnabled=" PowEnabled]
            [SP "PoWQueueRate=" PowQRate]
            [SP "PoWQueueBurst=" PowQBurt]
            1*(SP "Port=" VirtPort ["," Target])
            *(SP "ClientAuth=" ClientName [":" ClientBlob]) CRLF
            *(SP "ClientAuthV3=" V3Key) CRLF

    KeyType =
     "NEW"     / ; The server should generate a key of algorithm
KeyBlob
     "RSA1024" / ; The server should use the 1024 bit RSA key provided
                 in as KeyBlob (v2).
     "ED25519-V3"; The server should use the ed25519 v3 key provided
in as
                 KeyBlob (v3).

    KeyBlob =
     "BEST"    / ; The server should generate a key using the "best"
                 supported algorithm (KeyType == "NEW").
                 [As of 0.4.2.3-alpha, ED25519-V3 is used]
     "RSA1024" / ; The server should generate a 1024 bit RSA key
                 (KeyType == "NEW") (v2).
     "ED25519-V3"; The server should generate an ed25519 private key
                 (KeyType == "NEW") (v3).
     String      ; A serialized private key (without whitespace)

    Flag =
     "DiscardPK" / ; The server should not include the newly generated
                   private key as part of the response.
     "Detach"    / ; Do not associate the newly created Onion Service
                   to the current control connection.
     "BasicAuth" / ; Client authorization is required using the
"basic"
                   method (v2 only).
     "V3Auth"    / ; Version 3 client authorization is required (v3
only).

     "NonAnonymous" /; Add a non-anonymous Single Onion Service. Tor
                      checks this flag matches its configured hidden
                      service anonymity mode.
     "MaxStreamsCloseCircuit"; Close the circuit is the maximum
streams
                               allowed is reached.

    NumStreams = A value between 0 and 65535 which is used as the
maximum
                 streams that can be attached on a rendezvous circuit.
Setting
                 it to 0 means unlimited which is also the default
behavior.
```

Pluggable Transport Client Per-Connection Arguments

Certain PT transport protocols require that the client provides per-connection arguments when making outgoing connections. On the server side, this is handled by the “ARGS” optional argument as part of the “SMETHOD” message.

On the client side, arguments are passed via the authentication fields that are part of the SOCKS protocol.

First the “`<Key>=<Value>`” formatted arguments MUST be escaped, such that all backslash, equal sign, and semicolon characters are escaped with a backslash.

Second, all of the escaped are concatenated together.

Example:

shared-secret=rahasia;secrets-file=/tmp/blob

Lastly the arguments are transmitted when making the outgoing connection using the authentication mechanism specific to the SOCKS protocol version.

- In the case of SOCKS 4, the concatenated argument list is transmitted in the "USERID" field of the "CONNECT" request.

- In the case of SOCKS 5, the parent process must negotiate "Username/Password" authentication [RFC1929], and transmit the arguments encoded in the "UNAME" and "PASSWD" fields.

If the encoded argument list is less than 255 bytes in length, the “PLEN” field must be set to “1” and the “PASSWD” field must contain a single NUL character.

Anonymity Considerations

When designing and implementing a Pluggable Transport, care should be taken to preserve the privacy of clients and to avoid leaking personally identifying information.

Examples of client related considerations are:

- Not logging client IP addresses to disk.
- Not leaking DNS addresses except when necessary.
  - Ensuring that "TOR_PT_PROXY"'s "fail closed" behavior is implemented correctly.

Additionally, certain obfuscation mechanisms rely on information such as the server IP address/port being confidential, so clients also need to take care to preserve server side information confidential when applicable.

```
     C: HSFETCH v2-gezdgnbvgy3tqolbmjrwizlgm5ugs2tl
        SERVER=9695DFC35FFEB861329B9F1AB04C46397020CE31
     S: 250 OK

     C: HSFETCH ajkhdsfuygaesfaa
     S: 250 OK

     C: HSFETCH
 vww6ybal4bd7szmgncyruucpgfkqahzddi37ktceo3ah7ngmcopnpyyd
     S: 250 OK
```

[HSFETCH was added in Tor 0.2.7.1-alpha] [HS v3 support added 0.4.1.1-alpha]

ADD_ONION

The syntax is:

[DROPGUARDS was added in Tor 0.2.5.2-alpha.]

HSFETCH

The syntax is:

```
"HSFETCH" SP (HSAddress / "v" Version "-" DescId)
          *[SP "SERVER=" Server] CRLF

HSAddress = 16*Base32Character / 56*Base32Character
Version = "2" / "3"
DescId = 32*Base32Character
Server = LongName
```

This command launches hidden service descriptor fetch(es) for the given HSAddress or DescId.

HSAddress can be version 2 or version 3 addresses. DescIDs can only be version 2 IDs. Version 2 addresses consist of 16*Base32Character and version 3 addresses consist of 56*Base32Character.

If a DescId is specified, at least one Server MUST also be provided, otherwise a 512 error is returned. If no DescId and Server(s) are specified, it behaves like a normal Tor client descriptor fetch. If one or more Server are given, they are used instead triggering a fetch on each of them in parallel.

The caching behavior when fetching a descriptor using this command is identical to normal Tor client behavior.

Details on how to compute a descriptor id (DescId) can be found in rend-spec.txt section 1.3.

If any values are unrecognized, a 513 error is returned and the command is stopped. On success, Tor replies “250 OK” then Tor MUST eventually follow this with both a HS_DESC and HS_DESC_CONTENT events with the results. If SERVER is specified then events are emitted for each location.

Examples are:

References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L., "SOCKS Protocol Version 5", RFC 1928, March 1996.

[EXTORPORT] Kadianakis, G., Mathewson, N., "Extended ORPort and TransportControlPort", Tor Proposal 196, March 2012.

[RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, March 1996.

Acknowledgments

This specification draws heavily from prior versions done by Jacob Appelbaum, Nick Mathewson, and George Kadianakis.

```
"250 AUTHCHALLENGE"
        SP "SERVERHASH=" ServerHash
        SP "SERVERNONCE=" ServerNonce
        CRLF

ServerHash = 64*64HEXDIG
ServerNonce = 64*64HEXDIG
```

The ClientNonce, ServerHash, and ServerNonce values are encoded/decoded in the same way as the argument passed to the AUTHENTICATE command. ServerNonce MUST be 32 bytes long.

ServerHash is computed as:

```
    HMAC-SHA256("Tor safe cookie authentication server-to-controller
hash",
                CookieString | ClientNonce | ServerNonce)

  (with the HMAC key as its first argument)
```

After a controller sends a successful AUTHCHALLENGE command, the next command sent on the connection must be an AUTHENTICATE command, and the only authentication string which that AUTHENTICATE command will accept is:

```
    HMAC-SHA256("Tor safe cookie authentication controller-to-server
hash",
                CookieString | ClientNonce | ServerNonce)
```

[Unlike other commands besides AUTHENTICATE, AUTHCHALLENGE may be used (but only once!) before AUTHENTICATE.]

[AUTHCHALLENGE was added in Tor 0.2.3.13-alpha.]

DROPGUARDS

The syntax is:

```
    "DROPGUARDS" CRLF
```

Tells the server to drop all guard nodes. Do not invoke this command lightly; it can increase vulnerability to tracking attacks over time.

Tor replies with `250 OK` on success.

This command is intended to be used with the __OwningControllerProcess configuration option. A controller that starts a Tor process which the user cannot easily control or stop should 'own' that Tor process:

```
* When starting Tor, the controller should specify its PID in an
  __OwningControllerProcess on Tor's command line.  This will
  cause Tor to poll for the existence of a process with that PID,
  and exit if it does not find such a process.  (This is not a
  completely reliable way to detect whether the 'owning
  controller' is still running, but it should work well enough in
  most cases.)

* Once the controller has connected to Tor's control port, it
  should send the TAKEOWNERSHIP command along its control
  connection.  At this point, *both* the TAKEOWNERSHIP command and
  the __OwningControllerProcess option are in effect: Tor will
  exit when the control connection ends *and* Tor will exit if it
  detects that there is no process with the PID specified in the
  __OwningControllerProcess option.

* After the controller has sent the TAKEOWNERSHIP command, it
  should send "RESETCONF __OwningControllerProcess" along its
  control connection.  This will cause Tor to stop polling for the
  existence of a process with its owning controller's PID; Tor
  will still exit when the control connection ends.
```

[TAKEOWNERSHIP was added in Tor 0.2.2.28-beta.]

AUTHCHALLENGE

The syntax is:

```
"AUTHCHALLENGE" SP "SAFECOOKIE"
                SP ClientNonce
                CRLF

ClientNonce = 2*HEXDIG / QuotedString
```

This command is used to begin the authentication routine for the SAFECOOKIE method of authentication.

If the server accepts the command, the server reply format is:

Appendix A: Example Client Pluggable Transport Session

Environment variables:

TOR_PT_MANAGED_TRANSPORT_VER=1 TOR_PT_STATE_LOCATION=/var/lib/tor/pt_state/ TOR_PT_EXIT_ON_STDIN_CLOSE=1
TOR_PT_PROXY=socks5://127.0.0.1:8001
TOR_PT_CLIENT_TRANSPORTS=obfs3,obfs4

Messages the PT Proxy writes to stdin:

VERSION 1 PROXY DONE CMETHOD obfs3 socks5 127.0.0.1:32525 CMETHOD obfs4 socks5 127.0.0.1:37347 CMETHODS DONE

Appendix B: Example Server Pluggable Transport Session

Environment variables:

TOR_PT_MANAGED_TRANSPORT_VER=1 TOR_PT_STATE_LOCATION=/var/lib/tor/pt_state TOR_PT_EXIT_ON_STDIN_CLOSE=1 TOR_PT_SERVER_TRANSPORTS=obfs3,obfs4 TOR_PT_SERVER_BINDADDR=obfs3-198.51.100.1:1984

Messages the PT Proxy writes to stdin:

VERSION 1 SMETHOD obfs3 198.51.100.1:1984 SMETHOD obfs4 198.51.100.1:43734 ARGS:cert=HszPy3vWfjsESCEOo9ZBkRv6zQ/1mGHzc8arF0y2SpwFr3WhsMu8rK0zyaoyERfbz3ddFw,iat-mode=0 SMETHODS DONE

The COOKIE authentication method exposes the user running a controller to an unintended information disclosure attack whenever the controller has greater filesystem read access than the process that it has connected to. (Note that a controller may connect to a process other than Tor.) It is almost never safe to use, even if the controller's user has explicitly specified which filename to read an authentication cookie from. For this reason, the COOKIE authentication method has been deprecated and will be removed from a future version of Tor.

The VERSION line contains the Tor version.

[Unlike other commands besides AUTHENTICATE, PROTOCOLINFO may be used (but only once!) before AUTHENTICATE.]

[PROTOCOLINFO was not supported before Tor 0.2.0.5-alpha.]

LOADCONF

The syntax is:

```
"+LOADCONF" CRLF ConfigText CRLF "." CRLF
```

This command allows a controller to upload the text of a config file to Tor over the control port. This config file is then loaded as if it had been read from disk.

[LOADCONF was added in Tor 0.2.1.1-alpha.]

TAKEOWNERSHIP

The syntax is:

```
"TAKEOWNERSHIP" CRLF
```

This command instructs Tor to shut down when this control connection is closed. This command affects each control connection that sends it independently; if multiple control connections send the TAKEOWNERSHIP command to a Tor instance, Tor will shut down when any of those connections closes.

(As of Tor 0.2.5.2-alpha, Tor does not wait a while for circuits to close when shutting down because of an exiting controller. If you want to ensure a clean shutdown–and you should!–then send "SIGNAL SHUTDOWN" and wait for the Tor process to close.)

```
    "250-PROTOCOLINFO" SP PIVERSION CRLF \*InfoLine "250 OK" CRLF

    InfoLine = AuthLine / VersionLine / OtherLine

    AuthLine = "250-AUTH" SP "METHODS=" AuthMethod *("," AuthMethod)
                       *(SP "COOKIEFILE=" AuthCookieFile) CRLF
    VersionLine = "250-VERSION" SP "Tor=" TorVersion OptArguments CRLF

    AuthMethod =
      "NULL"           / ; No authentication is required
      "HASHEDPASSWORD" / ; A controller must supply the original
password
      "COOKIE"         / ; ... or supply the contents of a cookie file
      "SAFECOOKIE"       ; ... or prove knowledge of a cookie file's
contents

    AuthCookieFile = QuotedString
    TorVersion = QuotedString

    OtherLine = "250-" Keyword OptArguments CRLF

    PIVERSION: 1*DIGIT
```

This command tells the controller what kinds of authentication are supported.

Tor MAY give its InfoLines in any order; controllers MUST ignore InfoLines with keywords they do not recognize. Controllers MUST ignore extraneous data on any InfoLine.

PIVERSION is there in case we drastically change the syntax one day. For now it should always be “1”. Controllers MAY provide a list of the protocolinfo versions they support; Tor MAY select a version that the controller does not support.

AuthMethod is used to specify one or more control authentication methods that Tor currently accepts.

AuthCookieFile specifies the absolute path and filename of the authentication cookie that Tor is expecting and is provided iff the METHODS field contains the method “COOKIE” and/or “SAFECOOKIE”. Controllers MUST handle escape sequences inside this string.

All authentication cookies are 32 bytes long. Controllers MUST NOT use the contents of a non-32-byte-long file as an authentication cookie.

If the METHODS field contains the method “SAFECOOKIE”, every AuthCookieFile must contain the same authentication cookie.

TC: A Tor control protocol (Version 1)

Scope

This document describes an implementation-specific protocol that is used for other programs (such as frontend user-interfaces) to communicate with a locally running Tor process. It is not part of the Tor onion routing protocol.

This protocol replaces version 0 of TC, which is now deprecated. For reference, TC is described in “control-spec-v0.txt”. Implementors are recommended to avoid using TC directly, but instead to use a library that can easily be updated to use the newer protocol. (Version 0 is used by Tor versions 0.1.0.x; the protocol in this document only works with Tor versions in the 0.1.1.x series and later.)

```
      The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
      NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED",  "MAY", and
      "OPTIONAL" in this document are to be interpreted as described
in
      RFC 2119.
```

Protocol outline

TC is a bidirectional message-based protocol. It assumes an underlying stream for communication between a controlling process (the “client” or “controller”) and a Tor process (or “server”). The stream may be implemented via TCP, TLS-over-TCP, a Unix-domain socket, or so on, but it must provide reliable in-order delivery. For security, the stream should not be accessible by untrusted parties.

In TC, the client and server send typed messages to each other over the underlying stream. The client sends “commands” and the server sends “replies”.

By default, all messages from the server are in response to messages from the client. Some client requests, however, will cause the server to send messages to the client indefinitely far into the future. Such “asynchronous” replies are marked as such.

Servers respond to messages in the order messages are received.

Forward-compatibility

This is an evolving protocol; new client and server behavior will be allowed in future versions. To allow new backward-compatible behavior on behalf of the client, we may add new commands and allow existing commands to take new arguments in future versions. To allow new backward-compatible server behavior, we note various places below where servers speaking a future version of this protocol may insert new data, and note that clients should/must “tolerate” unexpected elements in these places. There are two ways that we do this:

- Adding a new field to a message:

EXTENDED_EVENTS

Same as passing ‘EXTENDED’ to SETEVENTS; this is the preferred way to request the extended event syntax.

This feature was first introduced in 0.1.2.3-alpha. It is always-on and part of the protocol in Tor 0.2.2.1-alpha and later.

VERBOSE_NAMES

Replaces ServerID with LongName in events and GETINFO results. LongName provides a Fingerprint for all routers, an indication of Named status, and a Nickname if one is known. LongName is strictly more informative than ServerID, which only provides either a Fingerprint or a Nickname.

This feature was first introduced in 0.1.2.2-alpha. It is always-on and part of the protocol in Tor 0.2.2.1-alpha and later.

RESOLVE

The syntax is

```
"RESOLVE" *Option *Address CRLF
Option = "mode=reverse"
Address = a hostname or IPv4 address
```

This command launches a remote hostname lookup request for every specified request (or reverse lookup if “mode=reverse” is specified). Note that the request is done in the background: to see the answers, your controller will need to listen for ADDRMAP events; see 4.1.7 below.

[Added in Tor 0.2.0.3-alpha]

PROTOCOLINFO

The syntax is:

```
"PROTOCOLINFO" *(SP PIVERSION) CRLF
```

The server reply format is:

```
"CLOSECIRCUIT" SP CircuitID *(SP Flag) CRLF
Flag = "IfUnused"
```

Tells the server to close the specified circuit. If “IfUnused” is provided, do not close the circuit unless it is unused.

Other flags may be defined in the future; Tor SHOULD ignore unrecognized flags.

Tor replies with `250 OK` on success, or a `512` if there aren’t enough arguments, or a `552` if it doesn’t recognize the CircuitID.

QUIT

Tells the server to hang up on this controller connection. This command can be used before authenticating.

USEFEATURE

Adding additional features to the control protocol sometimes will break backwards compatibility. Initially such features are added into Tor and disabled by default. USEFEATURE can enable these additional features.

The syntax is:

```
"USEFEATURE" *(SP FeatureName) CRLF
FeatureName = 1*(ALPHA / DIGIT / "_" / "-")
```

Feature names are case-insensitive.

Once enabled, a feature stays enabled for the duration of the connection to the controller. A new connection to the controller must be opened to disable an enabled feature.

Features are a forward-compatibility mechanism; each feature will eventually become a standard part of the control protocol. Once a feature becomes part of the protocol, it is always-on. Each feature documents the version it was introduced as a feature and the version in which it became part of the protocol.

Tor will ignore a request to use any feature that is always-on. Tor will give a `552` error in response to an unrecognized feature.

```
    For example, we might say "This message has three space-separated
    fields; clients MUST tolerate more fields."  This means that a
    client MUST NOT crash or otherwise fail to parse the message or
    other subsequent messages when there are more than three fields,
and
    that it SHOULD function at least as well when more fields are
    provided as it does when it only gets the fields it accepts.  The
    most obvious way to do this is by ignoring additional fields; the
    next-most-obvious way is to report additional fields verbatim to
the
    user, perhaps as part of an expert UI.
```

- Adding a new possible value to a list of alternatives:

```
    For example, we might say "This field will be OPEN, CLOSED, or
    CONNECTED.  Clients MUST tolerate unexpected values."  This means
    that a client MUST NOT crash or otherwise fail to parse the
message
    or other subsequent messages when there are unexpected values, and
    that it SHOULD try to handle the rest of the message as well as it
    can.  The most obvious way to do this is by pretending that each
    list of alternatives has an additional "unrecognized value"
element,
    and mapping any unrecognized values to that element; the
    next-most-obvious way is to create a separate "unrecognized value"
    element for each unrecognized value.

    Clients SHOULD NOT "tolerate" unrecognized alternatives by
    pretending that the message containing them is absent.  For
example,
    a stream closed for an unrecognized reason is nevertheless closed,
    and should be reported as such.

    (If some list of alternatives is given, and there isn't an
explicit
    statement that clients must tolerate unexpected values, clients
still
    must tolerate unexpected values. The only exception would be if
there
    were an explicit statement that no future values will ever be
added.)
```

Message format

Description format

The message formats listed below use ABNF as described in RFC 2234. The protocol itself is loosely based on SMTP (see RFC 2821).

We use the following nonterminals from RFC 2822: atom, qcontent

We define the following general-use nonterminals:

```
QuotedString = DQUOTE *qcontent DQUOTE
```

There are explicitly no limits on line length. All 8-bit characters are permitted unless explicitly disallowed. In QuotedStrings, backslashes and quotes must be escaped; other characters need not be escaped.

Wherever CRLF is specified to be accepted from the controller, Tor MAY also accept LF. Tor, however, MUST NOT generate LF instead of CRLF. Controllers SHOULD always send CRLF.

Notes on an escaping bug

```
CString = DQUOTE *qcontent DQUOTE
```

Note that although these nonterminals have the same grammar, they are interpreted differently. In a QuotedString, a backslash followed by any character represents that character. But in a CString, the escapes “\n”, “\t”, “\r”, and the octal escapes “\0” … “\377” represent newline, tab, carriage return, and the 256 possible octet values respectively.

The use of CString in this document reflects a bug in Tor; they should have been QuotedString instead. In the future, they may migrate to use QuotedString instead. If they do, the QuotedString implementation will never place a backslash before a “n”, “t”, “r”, or digit, to ensure that old controllers don’t get confused.

For future-proofing, controller implementors MAY use the following rules to be compatible with buggy Tor implementations and with future ones that implement the spec as intended:

The descriptor, when parsed, must contain a number of well-specified fields, including fields for its nickname and identity.

If there is an error in parsing the descriptor, the server must send a `554 Invalid descriptor` reply. If the descriptor is well-formed but the server chooses not to add it, it must reply with a `251` message whose body explains why the server was not added. If the descriptor is added, Tor replies with `250 OK`.

REDIRECTSTREAM

Sent from the client to the server. The syntax is:

```
"REDIRECTSTREAM" SP StreamID SP Address [SP Port] CRLF
```

Tells the server to change the exit address on the specified stream. If Port is specified, changes the destination port as well. No remapping is performed on the new provided address.

To be sure that the modified address will be used, this event must be sent after a new stream event is received, and before attaching this stream to a circuit.

Tor replies with `250 OK` on success.

CLOSESTREAM

Sent from the client to the server. The syntax is:

```
"CLOSESTREAM" SP StreamID SP Reason *(SP Flag) CRLF
```

Tells the server to close the specified stream. The reason should be one of the Tor RELAY_END reasons given in tor-spec.txt, as a decimal. Flags is not used currently; Tor servers SHOULD ignore unrecognized flags. Tor may hold the stream open for a while to flush any data that is pending.

Tor replies with `250 OK` on success, or a `512` if there aren’t enough arguments, or a `552` if it doesn’t recognize the StreamID or reason.

CLOSECIRCUIT

The syntax is:

and multiple streams may share the same circuit. Streams can only be attached to completed circuits (that is, circuits that have sent a circuit status 'BUILT' event or are listed as built in a GETINFO circuit-status request).

If the circuit ID is 0, responsibility for attaching the given stream is returned to Tor.

If HOP=HopNum is specified, Tor will choose the HopNumth hop in the circuit as the exit node, rather than the last node in the circuit. Hops are 1-indexed; generally, it is not permitted to attach to hop 1.

Tor responds with `250 OK` if it can attach the stream, `552` if the circuit or stream didn't exist, `555` if the stream isn't in an appropriate state to be attached (e.g. it's already open), or `551` if the stream couldn't be attached for another reason.

{Implementation note: Tor will close unattached streams by itself, roughly two minutes after they are born. Let the developers know if that turns out to be a problem.}

{Implementation note: By default, Tor automatically attaches streams to circuits itself, unless the configuration variable "__LeaveStreamsUnattached" is set to "1". Attempting to attach streams via TC when "__LeaveStreamsUnattached" is false may cause a race between Tor and the controller, as both attempt to attach streams to circuits.}

{Implementation note: You can try to attachstream to a stream that has already sent a connect or resolve request but hasn't succeeded yet, in which case Tor will detach the stream from its current circuit before proceeding with the new attach request.}

POSTDESCRIPTOR

Sent from the client to the server. The syntax is:

```
"+POSTDESCRIPTOR" [SP "purpose=" Purpose] [SP "cache=" Cache]
                  CRLF Descriptor CRLF "." CRLF
```

This message informs the server about a new descriptor. If Purpose is specified, it must be either "general", "controller", or "bridge", else we return a `552` error. The default is "general".

If Cache is specified, it must be either "no" or "yes", else we return a `552` error. If Cache is not specified, Tor will decide for itself whether it wants to cache the descriptor, and controllers must not rely on its choice.

```
    Read \n \t \r and \0 ... \377 as C escapes.
    Treat a backslash followed by any other character as that
character.
```

Currently, many of the QuotedString instances below that Tor outputs are in fact CStrings. We intend to fix this in future versions of Tor, and document which ones were broken. (See bugtracker ticket #14555 for a bit more information.)

Note that this bug exists only in strings generated by Tor for the Tor controller; Tor should parse input QuotedStrings from the controller correctly.

Commands from controller to Tor

```
    Command = Keyword OptArguments CRLF / "+" Keyword OptArguments
CRLF CmdData
    Keyword = 1*ALPHA
    OptArguments = [ SP *(SP / VCHAR) ]
```

A command is either a single line containing a Keyword and arguments, or a multiline command whose initial keyword begins with +, and whose data section ends with a single "." on a line of its own. (We use a special character to distinguish multiline commands so that Tor can correctly parse multi-line commands that it does not recognize.) Specific commands and their arguments are described below in section 3.

Replies from Tor to the controller

```
    Reply = SyncReply / AsyncReply
    SyncReply = *(MidReplyLine / DataReplyLine) EndReplyLine
    AsyncReply = *(MidReplyLine / DataReplyLine) EndReplyLine

    MidReplyLine = StatusCode "-" ReplyLine
    DataReplyLine = StatusCode "+" ReplyLine CmdData
    EndReplyLine = StatusCode SP ReplyLine
    ReplyLine = [ReplyText] CRLF
    ReplyText = XXXX
    StatusCode = 3DIGIT
```

Unless specified otherwise, multiple lines in a single reply from Tor to the controller are guaranteed to share the same status code. Specific replies are mentioned below in section 3, and described more fully in section 4.

[Compatibility note: versions of Tor before 0.2.0.3-alpha sometimes generate AsyncReplies of the form `*(MidReplyLine / DataReplyLine)`. This is incorrect, but controllers that need to work with these versions of Tor should be prepared to get multi-line AsyncReplies with the final line (usually `650 OK`) omitted.]

General-use tokens

CRLF means, "the ASCII Carriage Return character (decimal value 13) followed by the ASCII Linefeed character (decimal value 10)."

```
CRLF = CR LF
```

How a controller tells Tor about a particular OR. There are four possible formats:

- `$Fingerprint` – The router whose identity key hashes to the fingerprint. This is the preferred way to refer to an OR.
- `$Fingerprint~Nickname` – The router whose identity key hashes to the given fingerprint, but only if the router has the given nickname.
- `$Fingerprint=Nickname` – The router whose identity key hashes to the given fingerprint, but only if the router is Named and has the given nickname.
- `Nickname` – The Named router with the given nickname, or, if no such router exists, any router whose nickname matches the one given. This is not a safe way to refer to routers, since Named status could under some circumstances change over time.

The tokens that implement the above follow:

```
ServerSpec = LongName / Nickname
LongName   = Fingerprint [ "~" Nickname ]
```

For tors older than 0.3.1.3-alpha, LongName may have included an equal sign ("=") in lieu of a tilde ("~"). The presence of an equal sign denoted that the OR possessed the "Named" flag:

```
LongName   = Fingerprint [ ( "=" / "~" ) Nickname ]

Fingerprint = "$" 40*HEXDIG
NicknameChar = "a"-"z" / "A"-"Z" / "0" - "9"
Nickname = 1*19 NicknameChar
```

What follows is an outdated way to refer to ORs. Feature VERBOSE_NAMES replaces ServerID with LongName in events and GETINFO results.

If the CircuitID is 0, the controller has the option of providing a path for Tor to use to build the circuit. If it does not provide a path, Tor will select one automatically from high capacity nodes according to path-spec.txt.

If CircuitID is 0 and "purpose=" is specified, then the circuit's purpose is set. Two choices are recognized: "general" and "controller". If not specified, circuits are created as "general".

If the request is successful, the server sends a reply containing a message body consisting of the CircuitID of the (maybe newly created) circuit. The syntax is:

```
"250" SP "EXTENDED" SP CircuitID CRLF
```

SETCIRCUITPURPOSE

Sent from the client to the server. The format is:

```
"SETCIRCUITPURPOSE" SP CircuitID SP "purpose=" Purpose CRLF
```

This changes the circuit's purpose. See EXTENDCIRCUIT above for details.

SETROUTERPURPOSE

Sent from the client to the server. The format is:

```
"SETROUTERPURPOSE" SP NicknameOrKey SP Purpose CRLF
```

This changes the descriptor's purpose. See +POSTDESCRIPTOR below for details.

NOTE: This command was disabled and made obsolete as of Tor 0.2.0.8-alpha. It doesn't exist anymore, and is listed here only for historical interest.

ATTACHSTREAM

Sent from the client to the server. The syntax is:

```
"ATTACHSTREAM" SP StreamID SP CircuitID [SP "HOP=" HopNum] CRLF
```

This message informs the server that the specified stream should be associated with the specified circuit. Each stream may be associated with at most one circuit,

```
      assigned.  (Introduced in 0.4.5.1-alpha)

    "stats/tap/requested"
    "stats/tap/assigned"
      The TAP circuit onion handshake rephist values which are
requested or
      assigned.  (Introduced in 0.4.5.1-alpha)

    "config-can-saveconf"
      0 or 1, depending on whether it is possible to use SAVECONF
without the
      FORCE flag. (Introduced in 0.3.1.1-alpha.)

    "limits/max-mem-in-queues"
      The amount of memory that Tor's out-of-memory checker will allow
      Tor to allocate (in places it can see) before it starts freeing
memory
      and killing circuits. See the MaxMemInQueues option for more
      details. Unlike the option, this value reflects Tor's actual
limit, and
      may be adjusted depending on the available system memory rather
than on
      the MaxMemInQueues option. (Introduced in 0.2.5.4-alpha)
```

Example:

```
C: GETINFO version desc/name/moria1
S: 250+desc/name/moria=
S: [Descriptor for moria]
S: .
S: 250-version=Tor 0.1.1.0-alpha-cvs
S: 250 OK
```

EXTENDCIRCUIT

Sent from the client to the server. The format is:

```
"EXTENDCIRCUIT" SP CircuitID
                [SP ServerSpec *("," ServerSpec)]
                [SP "purpose=" Purpose] CRLF
```

This request takes one of two forms: either the CircuitID is zero, in which case it is a request for the server to build a new circuit, or the CircuitID is nonzero, in which case it is a request for the server to extend an existing circuit with that ID according to the specified path.

VERBOSE_NAMES can be enabled starting in Tor version 0.1.2.2-alpha and it is always-on in 0.2.2.1-alpha and later.

```
ServerID = Nickname / Fingerprint
```

Unique identifiers for streams or circuits. Currently, Tor only uses digits, but this may change:

```
StreamID = 1*16 IDChar
CircuitID = 1*16 IDChar
ConnID = 1*16 IDChar
QueueID = 1*16 IDChar
IDChar = ALPHA / DIGIT
```

```
Address = ip4-address / ip6-address / hostname   (XXXX Define
these)
```

A “CmdData” section is a sequence of octets concluded by the terminating sequence `CRLF "." CRLF`. The terminating sequence may not appear in the body of the data. Leading periods on lines in the data are escaped with an additional leading period as in RFC 2821 section 4.5.2.

```
CmdData = *DataLine "." CRLF
DataLine = CRLF / "." 1*LineItem CRLF / NonDotItem *LineItem CRLF
LineItem = NonCR / 1*CR NonCRLF
NonDotItem = NonDotCR / 1*CR NonCRLF
```

ISOTime, ISOTime2, and ISOTime2Frac are time formats as specified in ISO8601.

- example ISOTime: “2012-01-11 12:15:33”
- example ISOTime2: “2012-01-11T12:15:33”
- example ISOTime2Frac: “2012-01-11T12:15:33.51”

```
IsoDatePart = 4*DIGIT "-" 2*DIGIT "-" 2*DIGIT
IsoTimePart = 2*DIGIT ":" 2*DIGIT ":" 2*DIGIT
ISOTime  = IsoDatePart " " IsoTimePart
ISOTime2 = IsoDatePart "T" IsoTimePart
ISOTime2Frac = IsoTime2 [ "." 1*DIGIT ]
```

Numbers

```
LeadingDigit = "1" / ... / "9"
UInt = LeadingDigit *Digit
```

Commands

All commands are case-insensitive, but most keywords are case-sensitive.

SETCONF

Change the value of one or more configuration variables. The syntax is:

```
"SETCONF" 1*(SP keyword ["=" value]) CRLF
value = String / QuotedString
```

Tor behaves as though it had just read each of the key-value pairs from its configuration file. Keywords with no corresponding values have their configuration values reset to 0 or NULL (use RESETCONF if you want to set it back to its default). SETCONF is all-or-nothing: if there is an error in any of the configuration settings, Tor sets none of them.

Tor responds with a “250 OK” reply on success. If some of the listed keywords can’t be found, Tor replies with a “552 Unrecognized option” message. Otherwise, Tor responds with a “513 syntax error in configuration values” reply on syntax error, or a “553 impossible configuration setting” reply on a semantic error.

Some configuration options (e.g. “Bridge”) take multiple values. Also, some configuration keys (e.g. for hidden services and for entry guard lists) form a context-sensitive group where order matters (see GETCONF below). In these cases, setting *any* of the options in a SETCONF command is taken to reset all of the others. For example, if two ORListenAddress values are configured, and a SETCONF command arrives containing a single ORListenAddress value, the new command’s value replaces the two old values.

Sometimes it is not possible to change configuration options solely by issuing a series of SETCONF commands, because the value of one of the configuration options depends on the value of another which has not yet been set. Such situations can be overcome by setting multiple configuration options with a single SETCONF command (e.g. SETCONF ORPort=443 ORListenAddress=9001).

```
      "downloads/cert/fp/<Fingerprint>/<SKDigest>"
        A SerializedDownloadStatus for the certificate for the
identity
        digest <Fingerprint> returned by the downloads/cert/fps key
and signing
        key digest <SKDigest> returned by the downloads/cert/fp/
<Fingerprint>/
        sks key.

      "downloads/desc/descs"
        A newline-separated list of hex-encoded router descriptor
digests
        [note, not identity digests - the Tor process may not have
seen them
        yet while downloading router descriptors].  If the Tor process
is not
        using a NS-flavored consensus, a 551 error is returned.

      "downloads/desc/<Digest>"
        A SerializedDownloadStatus for the router descriptor with
digest
        <Digest> as returned by the downloads/desc/descs key.  If the
Tor
        process is not using a NS-flavored consensus, a 551 error is
returned.

      "downloads/bridge/bridges"
        A newline-separated list of hex-encoded bridge identity
digests.  If
        the Tor process is not using bridges, a 551 error is returned.

      "downloads/bridge/<Digest>"
        A SerializedDownloadStatus for the bridge descriptor with
identity
        digest <Digest> as returned by the downloads/bridge/bridges
key.  If
        the Tor process is not using bridges, a 551 error is returned.

    "sr/current"
    "sr/previous"
      The current or previous shared random value, as received in the
      consensus, base-64 encoded.  An empty value means that either
      the consensus has no shared random value, or Tor has no
consensus.

    "current-time/local"
    "current-time/utc"
      The current system or UTC time, as returned by the system, in
ISOTime2
      format.  (Introduced in 0.3.4.1-alpha.)

    "stats/ntor/requested"
    "stats/ntor/assigned"
      The NTor circuit onion handshake rephist values which are
requested or
```

```
      The optional last two lines must be present if DownloadBackoff
is
      "DL_SCHED_RANDOM_EXPONENTIAL" and must be absent if
DownloadBackoff
      is "DL_SCHED_DETERMINISTIC".

      In detail, the keys supported are:

      "downloads/networkstatus/ns"
        The SerializedDownloadStatus for the NS-flavored consensus for
        whichever bootstrap state Tor is currently in.

      "downloads/networkstatus/ns/bootstrap"
        The SerializedDownloadStatus for the NS-flavored consensus at
        bootstrap time, regardless of whether we are currently
bootstrapping.

      "downloads/networkstatus/ns/running"

        The SerializedDownloadStatus for the NS-flavored consensus
when
        running, regardless of whether we are currently bootstrapping.

      "downloads/networkstatus/microdesc"
        The SerializedDownloadStatus for the microdesc-flavored
consensus for
        whichever bootstrap state Tor is currently in.

      "downloads/networkstatus/microdesc/bootstrap"
        The SerializedDownloadStatus for the microdesc-flavored
consensus at
        bootstrap time, regardless of whether we are currently
bootstrapping.

      "downloads/networkstatus/microdesc/running"
        The SerializedDownloadStatus for the microdesc-flavored
consensus when
        running, regardless of whether we are currently bootstrapping.

      "downloads/cert/fps"

        A newline-separated list of hex-encoded digests for authority
        certificates for which we have download status available.

      "downloads/cert/fp/<Fingerprint>"
        A SerializedDownloadStatus for the default certificate for the
        identity digest <Fingerprint> returned by the downloads/cert/
fps key.

      "downloads/cert/fp/<Fingerprint>/sks"
        A newline-separated list of hex-encoded signing key digests
for the
        authority identity digest <Fingerprint> returned by the
        downloads/cert/fps key.
```

RESETCONF

Remove all settings for a given configuration option entirely, assign its default value (if any), and then assign the String provided. Typically the String is left empty, to simply set an option back to its default. The syntax is:

```
"RESETCONF" 1*(SP keyword ["=" String]) CRLF
```

Otherwise it behaves like SETCONF above.

GETCONF

Request the value of zero or more configuration variable(s). The syntax is:

```
"GETCONF" *(SP keyword) CRLF
```

If all of the listed keywords exist in the Tor configuration, Tor replies with a series of reply lines of the form:

```
250 keyword=value
```

If any option is set to a ‘default’ value semantically different from an empty string, Tor may reply with a reply line of the form:

```
250 keyword
```

Value may be a raw value or a quoted string. Tor will try to use unquoted values except when the value could be misinterpreted through not being quoted. (Right now, Tor supports no such misinterpretable values for configuration options.)

If some of the listed keywords can’t be found, Tor replies with a `552 unknown configuration keyword` message.

If an option appears multiple times in the configuration, all of its key-value pairs are returned in order.

If no keywords were provided, Tor responds with `250 OK` message.

Some options are context-sensitive, and depend on other options with different keywords. These cannot be fetched directly. Currently there is only one such option: clients should use the “HiddenServiceOptions” virtual keyword to get all

HiddenServiceDir, HiddenServicePort, HiddenServiceVersion, and HiddenserviceAuthorizeClient option settings.

SETEVENTS

Request the server to inform the client about interesting events. The syntax is:

```
"SETEVENTS" [SP "EXTENDED"] *(SP EventCode) CRLF

EventCode = 1*(ALPHA / "_")  (see section 4.1.x for event types)
```

Any events *not* listed in the SETEVENTS line are turned off; thus, sending SETEVENTS with an empty body turns off all event reporting.

The server responds with a `250 OK` reply on success, and a `552 Unrecognized event` reply if one of the event codes isn't recognized. (On error, the list of active event codes isn't changed.)

If the flag string "EXTENDED" is provided, Tor may provide extra information with events for this connection; see 4.1 for more information. NOTE: All events on a given connection will be provided in extended format, or none. NOTE: "EXTENDED" was first supported in Tor 0.1.1.9-alpha; it is always-on in Tor 0.2.2.1-alpha and later.

Each event is described in more detail in Section 4.1.

AUTHENTICATE

Sent from the client to the server. The syntax is:

```
"AUTHENTICATE" [ SP 1*HEXDIG / QuotedString ] CRLF
```

This command is used to authenticate to the server. The provided string is one of the following:

```
        HSAddress
      [New in Tor 0.2.7.1-alpha.]
      [HS v3 support added 0.3.3.1-alpha]

    "network-liveness"
      The string "up" or "down", indicating whether we currently
believe the
      network is reachable.

    "downloads/"
      The keys under downloads/ are used to query download statuses;
they all
      return either a sequence of newline-terminated hex encoded
digests, or
      a "serialized download status" as follows:

       SerializedDownloadStatus =
         -- when do we plan to next attempt to download this object?
         "next-attempt-at" SP ISOTime CRLF
         -- how many times have we failed since the last success?
         "n-download-failures" SP UInt CRLF
         -- how many times have we tried to download this?
         "n-download-attempts" SP UInt CRLF
         -- according to which schedule rule will we download this?
         "schedule" SP DownloadSchedule CRLF
         -- do we want to fetch this from an authority, or will any
cache do?
         "want-authority" SP DownloadWantAuthority CRLF
         -- do we increase our download delay whenever we fail to
fetch this,
         -- or whenever we attempt fetching this?
         "increment-on" SP DownloadIncrementOn CRLF
         -- do we increase the download schedule deterministically, or
at
         -- random?
         "backoff" SP DownloadBackoff CRLF
         [
           -- with an exponential backoff, where are we in the
schedule?
           "last-backoff-position" Uint CRLF
           -- with an exponential backoff, what was our last delay?
           "last-delay-used UInt CRLF
         ]

      where

      DownloadSchedule =
        "DL_SCHED_GENERIC" / "DL_SCHED_CONSENSUS" / "DL_SCHED_BRIDGE"
      DownloadWantAuthority =
        "DL_WANT_ANY_DIRSERVER" / "DL_WANT_AUTHORITY"
      DownloadIncrementOn =
        "DL_SCHED_INCREMENT_FAILURE" / "DL_SCHED_INCREMENT_ATTEMPT"
      DownloadBackoff =
        "DL_SCHED_DETERMINISTIC" / "DL_SCHED_RANDOM_EXPONENTIAL"
```

```
      [New in Tor 0.2.6.3-alpha]

     "consensus/valid-after"
     "consensus/fresh-until"
     "consensus/valid-until"
      Each of these produces an ISOTime describing part of the
lifetime of
      the current (valid, accepted) consensus that Tor has.
      [New in Tor 0.2.6.3-alpha]

    "hs/client/desc/id/<ADDR>"
      Prints the content of the hidden service descriptor
corresponding to
      the given <ADDR> which is an onion address without the ".onion"
part.
      The client's cache is queried to find the descriptor. The format
of
      the descriptor is described in section 1.3 of the rend-spec.txt
      document.

      If <ADDR> is unrecognized or if not found in the cache, a 551
error is
      returned.

      [New in Tor 0.2.7.1-alpha]
      [HS v3 support added 0.3.3.1-alpha]

    "hs/service/desc/id/<ADDR>"
      Prints the content of the hidden service descriptor
corresponding to
      the given <ADDR> which is an onion address without the ".onion"
part.
      The service's local descriptor cache is queried to find the
descriptor.
      The format of the descriptor is described in section 1.3 of the
      rend-spec.txt document.

      If <ADDR> is unrecognized or if not found in the cache, a 551
error is
      returned.

      [New in Tor 0.2.7.2-alpha]
      [HS v3 support added 0.3.3.1-alpha]

    "onions/current"
    "onions/detached"
      A newline-separated list of the Onion ("Hidden") Services
created
      via the "ADD_ONION" command. The 'current' key returns Onion
Services
      belonging to the current control connection. The 'detached' key
      returns Onion Services detached from the parent control
connection
      (as in, belonging to no control connection).
      The format of each line is:
```

```
        * (For the HASHEDPASSWORD authentication method; see 3.21)
          The original password represented as a QuotedString.

        * (For the COOKIE is authentication method; see 3.21)
          The contents of the cookie file, formatted in hexadecimal

        * (For the SAFECOOKIE authentication method; see 3.21)
          The HMAC based on the AUTHCHALLENGE message, in hexadecimal.
```

The server responds with `250 OK` on success or `515 Bad authentication` if the authentication cookie is incorrect. Tor closes the connection on an authentication failure.

The authentication token can be specified as either a quoted ASCII string, or as an unquoted hexadecimal encoding of that same string (to avoid escaping issues).

For information on how the implementation securely stores authentication information on disk, see section 5.1.

Before the client has authenticated, no command other than PROTOCOLINFO, AUTHCHALLENGE, AUTHENTICATE, or QUIT is valid. If the controller sends any other command, or sends a malformed command, or sends an unsuccessful AUTHENTICATE command, or sends PROTOCOLINFO or AUTHCHALLENGE more than once, Tor sends an error reply and closes the connection.

To prevent some cross-protocol attacks, the AUTHENTICATE command is still required even if all authentication methods in Tor are disabled. In this case, the controller should just send “AUTHENTICATE” CRLF.

(Versions of Tor before 0.1.2.16 and 0.2.0.4-alpha did not close the connection after an authentication failure.)

SAVECONF

Sent from the client to the server. The syntax is:

```
"SAVECONF" [SP "FORCE"] CRLF
```

Instructs the server to write out its config options into its torrc. Server returns `250 OK` if successful, or `551 Unable to write configuration to disk` if it can’t write the file or some other error occurs.

If the %include option is used on torrc, SAVECONF will not write the configuration to disk. If the flag string “FORCE” is provided, the configuration will be overwritten

even if %include is used. Using %include on defaults-torrc does not affect SAVECONF. (Introduced in 0.3.1.1-alpha.)

See also the “getinfo config-text” command, if the controller wants to write the torrc file itself.

See also the “getinfo config-can-saveconf” command, to tell if the FORCE flag will be required. (Also introduced in 0.3.1.1-alpha.)

SIGNAL

Sent from the client to the server. The syntax is:

```
"SIGNAL" SP Signal CRLF

 Signal = "RELOAD" / "SHUTDOWN" / "DUMP" / "DEBUG" / "HALT" /
          "HUP" / "INT" / "USR1" / "USR2" / "TERM" / "NEWNYM" /
          "CLEARDNSCACHE" / "HEARTBEAT" / "ACTIVE" / "DORMANT"
```

The meaning of the signals are:

```
      Listeners for transparent connections redirected by firewall,
such as
      pf or netfilter.

    "net/listeners/natd"

      Listeners for transparent connections redirected by natd.

    "net/listeners/dns"

      Listeners for a subset of DNS protocol that Tor network
supports.

    "net/listeners/control"

      Listeners for Tor control protocol, described herein.

    "net/listeners/extor"

      Listeners corresponding to Extended ORPorts for integration with
      pluggable transports. See proposals 180 and 196.

    "net/listeners/httptunnel"

      Listeners for onion proxy connections that leverage HTTP CONNECT
      tunnelling.

      [The extor and httptunnel lists were added in 0.3.2.12,
0.3.3.10, and
      0.3.4.6-rc.]

    "dir-usage"
      A newline-separated list of how many bytes we've served to
answer
      each type of directory request. The format of each line is:
         Keyword 1*SP Integer 1*SP Integer
      where the first integer is the number of bytes written, and the
second
      is the number of requests answered.

      [This feature was added in Tor 0.2.2.1-alpha, and removed in
       Tor 0.2.9.1-alpha. Even when it existed, it only provided
       useful output when the Tor client was built with either the
       INSTRUMENT_DOWNLOADS or RUNNING_DOXYGEN compile-time options.]

    "bw-event-cache"
      A space-separated summary of recent BW events in chronological
order
      from oldest to newest.  Each event is represented by a comma-
separated
      tuple of "R,W", R is the number of bytes read, and W is the
number of
      bytes written.  These entries each represent about one second's
worth
      of traffic.
```

```
      use this getinfo when they connect or attach to Tor to learn its
      current bootstrap state.
    "status/version/recommended"
      List of currently recommended versions.
    "status/version/current"
      Status of the current version. One of: new, old, unrecommended,
      recommended, new in series, obsolete, unknown.
    "status/clients-seen"
      A summary of which countries we've seen clients from recently,
      formatted the same as the CLIENTS_SEEN status event described in
      Section 4.1.14. This GETINFO option is currently available only
      for bridge relays.
    "status/fresh-relay-descs"
      Provides fresh server and extra-info descriptors for our relay.
Note
      this is *not* the latest descriptors we've published, but rather
what we
      would generate if we needed to make a new descriptor right now.

    "net/listeners/*"

      A quoted, space-separated list of the locations where Tor is
listening
      for connections of the specified type. These can contain IPv4
      network address...

        "127.0.0.1:9050" "127.0.0.1:9051"

      ... or local unix sockets...

        "unix:/home/my_user/.tor/socket"

      ... or IPv6 network addresses:

        "[2001:0db8:7000:0000:0000:dead:beef:1234]:9050"

      [New in Tor 0.2.2.26-beta.]

    "net/listeners/or"

      Listeners for OR connections. Talks Tor protocol as described in
      tor-spec.txt.

    "net/listeners/dir"

      Listeners for Tor directory protocol, as described in dir-
spec.txt.

    "net/listeners/socks"

      Listeners for onion proxy connections that talk SOCKS4/4a/5
protocol.

    "net/listeners/trans"
```

```
      RELOAD    -- Reload: reload config items.
      SHUTDOWN  -- Controlled shutdown: if server is an OP, exit
immediately.
                   If it's an OR, close listeners and exit after
                   ShutdownWaitLength seconds.
      DUMP      -- Dump stats: log information about open connections
and
                   circuits.
      DEBUG     -- Debug: switch all open logs to loglevel debug.
      HALT      -- Immediate shutdown: clean up and exit now.
      CLEARDNSCACHE -- Forget the client-side cached IPs for all
hostnames.
      NEWNYM    -- Switch to clean circuits, so new application
requests
                   don't share any circuits with old ones.  Also
clears
                   the client-side DNS cache.  (Tor MAY rate-limit its
                   response to this signal.)
      HEARTBEAT -- Make Tor dump an unscheduled Heartbeat message to
log.
      DORMANT   -- Tell Tor to become "dormant".  A dormant Tor will
                   try to avoid CPU and network usage until it
receives
                   user-initiated network request.  (Don't use this
                   on relays or hidden services yet!)
      ACTIVE    -- Tell Tor to stop being "dormant", as if it had
received
                   a user-initiated network request.
```

The server responds with `250 OK` if the signal is recognized (or simply closes the socket if it was asked to close immediately), or `552 Unrecognized signal` if the signal is unrecognized.

Note that not all of these signals have POSIX signal equivalents. The ones that do are as below. You may also use these POSIX names for the signal that have them.

```
      RELOAD: HUP
      SHUTDOWN: INT
      HALT: TERM
      DUMP: USR1
      DEBUG: USR2
```

[SIGNAL DORMANT and SIGNAL ACTIVE were added in 0.4.0.1-alpha.]

MAPADDRESS

Sent from the client to the server. The syntax is:

```
"MAPADDRESS" SP 1*(Address "=" Address SP) CRLF
```

The first address in each pair is an “original” address; the second is a “replacement” address. The client sends this message to the server in order to tell it that future SOCKS requests for connections to the original address should be replaced with connections to the specified replacement address. If the addresses are well-formed, and the server is able to fulfill the request, the server replies with a 250 message:

```
250-OldAddress1=NewAddress1
250 OldAddress2=NewAddress2
```

containing the source and destination addresses. If request is malformed, the server replies with `512 syntax error in command argument`. If the server can’t fulfill the request, it replies with `451 resource exhausted`.

The client may decline to provide a body for the original address, and instead send a special null address (“0.0.0.0” for IPv4, “::0” for IPv6, or “.” for hostname), signifying that the server should choose the original address itself, and return that address in the reply. The server should ensure that it returns an element of address space that is unlikely to be in actual use. If there is already an address mapped to the destination address, the server may reuse that mapping.

If the original address is already mapped to a different address, the old mapping is removed. If the original address and the destination address are the same, the server removes any mapping in place for the original address.

Example:

```
C: MAPADDRESS 1.2.3.4=torproject.org
S: 250 1.2.3.4=torproject.org

C: GETINFO address-mappings/control
S: 250-address-mappings/control=1.2.3.4 torproject.org NEVER
S: 250 OK

C: MAPADDRESS 1.2.3.4=1.2.3.4
S: 250 1.2.3.4=1.2.3.4

C: GETINFO address-mappings/control
S: 250-address-mappings/control=
S: 250 OK
```

```
    "dir/server/fp/<F>"
    "dir/server/fp/<F1>+<F2>+<F3>"
    "dir/server/d/<D>"
    "dir/server/d/<D1>+<D2>+<D3>"
    "dir/server/authority"
    "dir/server/all"
      A series of lines listing directory contents, provided according
to the
      specification for the URLs listed in Section 4.4 of dir-
spec.txt.  Note
      that Tor MUST NOT provide private information, such as
descriptors for
      routers not marked as general-purpose.  When asked for
'authority'
      information for which this Tor is not authoritative, Tor replies
with
      an empty string.

      Note that, as of Tor 0.2.3.3-alpha, Tor clients don't download
server
      descriptors anymore, but microdescriptors.  So, a "551 Servers
      unavailable" reply to all "GETINFO dir/server/*" requests is
actually
      correct.  If you have an old program which absolutely requires
server
      descriptors to work, try setting UseMicrodescriptors 0 or
      FetchUselessDescriptors 1 in your client's torrc.

    "status/circuit-established"
    "status/enough-dir-info"
    "status/good-server-descriptor"
    "status/accepted-server-descriptor"
    "status/..."
      These provide the current internal Tor values for various Tor
      states. See Section 4.1.10 for explanations. (Only a few of the
      status events are available as getinfo's currently. Let us know
if
      you want more exposed.)
    "status/reachability-succeeded/or"
      0 or 1, depending on whether we've found our ORPort reachable.
    "status/reachability-succeeded/dir"
      0 or 1, depending on whether we've found our DirPort reachable.
      1 if there is no DirPort, and therefore no need for a
reachability
      check.
    "status/reachability-succeeded"
      "OR=" ("0"/"1") SP "DIR=" ("0"/"1")
      Combines status/reachability-succeeded/*; controllers MUST
ignore
      unrecognized elements in this entry.
    "status/bootstrap-phase"
      Returns the most recent bootstrap phase status event
      sent. Specifically, it returns a string starting with either
      "NOTICE BOOTSTRAP ..." or "WARN BOOTSTRAP ...". Controllers
should
```

```
    "info/names"
      A series of lines listing the available GETINFO options.  Each
is of
      one of these forms:
         OptionName SP Documentation CRLF
         OptionPrefix SP Documentation CRLF
         OptionPrefix = OptionName "/*"
      The OptionPrefix form indicates a number of options beginning
with the
      prefix. So if "config/*" is listed, other options beginning with
      "config/" will work, but "config/*" itself is not an option.

    "events/names"
      A space-separated list of all the events supported by this
version of
      Tor's SETEVENTS.

    "features/names"
      A space-separated list of all the features supported by this
version
      of Tor's USEFEATURE.

    "signal/names"
      A space-separated list of all the values supported by the SIGNAL
      command.

    "ip-to-country/ipv4-available"
    "ip-to-country/ipv6-available"
      "1" if the relevant geoip or geoip6 database is present; "0"
otherwise.
      This field was added in Tor 0.3.2.1-alpha.

    "ip-to-country/*"
      Maps IP addresses to 2-letter country codes.  For example,
      "GETINFO ip-to-country/18.0.0.1" should give "US".

    "process/pid" -- Process id belonging to the main tor process.
    "process/uid" -- User id running the tor process, -1 if unknown
(this is
     unimplemented on Windows, returning -1).
    "process/user" -- Username under which the tor process is running,
     providing an empty string if none exists (this is unimplemented
on
     Windows, returning an empty string).
    "process/descriptor-limit" -- Upper bound on the file descriptor
limit, -1
     if unknown

    "dir/status-vote/current/consensus" [added in Tor 0.2.1.6-alpha]
    "dir/status-vote/current/consensus-microdesc" [added in Tor
0.4.3.1-alpha]
    "dir/status/authority"
    "dir/status/fp/<F>"
    "dir/status/fp/<F1>+<F2>+<F3>"
    "dir/status/all"
```

Note

This feature is designed to be used to help Tor-ify applications that need to use SOCKS4 or hostname-less SOCKS5. There are three approaches to doing this:

1. Somehow make them use SOCKS4a or SOCKS5-with-hostnames instead.
2. Use tor-resolve (or another interface to Tor's resolve-over-SOCKS feature) to resolve the hostname remotely. This doesn't work with special addresses like x.onion or x.y.exit.
3. Use MAPADDRESS to map an IP address to the desired hostname, and then arrange to fool the application into thinking that the hostname has resolved to that IP.

This functionality is designed to help implement the 3rd approach.

Mappings set by the controller last until the Tor process exits: they never expire. If the controller wants the mapping to last only a certain time, then it must explicitly un-map the address when that time has elapsed.

MapAddress replies MAY contain mixed status codes.

Example:

```
C: MAPADDRESS xxx=@@@ 0.0.0.0=bogus1.google.com
S: 512-syntax error: invalid address '@@@'
S: 250 127.199.80.246=bogus1.google.com
```

GETINFO

Sent from the client to the server. The syntax is as for GETCONF:

```
"GETINFO" 1*(SP keyword) CRLF
```

Unlike GETCONF, this message is used for data that are not stored in the Tor configuration file, and that may be longer than a single line. On success, one ReplyLine is sent for each requested value, followed by a final 250 OK ReplyLine. If a value fits on a single line, the format is:

```
250-keyword=value
```

If a value must be split over multiple lines, the format is:

```
250+keyword=
value
.
```

The server sends a 551 or 552 error on failure.

Recognized keys and their values include:

```
    "traffic/written" -- Total bytes written (uploaded).

    "uptime" -- Uptime of the Tor daemon (in seconds).  Added in
       0.3.5.1-alpha.

    "accounting/enabled"
    "accounting/hibernating"
    "accounting/bytes"
    "accounting/bytes-left"
    "accounting/interval-start"
    "accounting/interval-wake"
    "accounting/interval-end"
      Information about accounting status.  If accounting is enabled,
      "enabled" is 1; otherwise it is 0.  The "hibernating" field is
"hard"
      if we are accepting no data; "soft" if we're accepting no new
      connections, and "awake" if we're not hibernating at all.  The
"bytes"
      and "bytes-left" fields contain (read-bytes SP write-bytes), for
the
      start and the rest of the interval respectively.  The 'interval-
start'
      and 'interval-end' fields are the borders of the current
interval; the
      'interval-wake' field is the time within the current interval
(if any)
      where we plan[ned] to start being active. The times are UTC.

    "config/names"
      A series of lines listing the available configuration options.
Each is
      of the form:
         OptionName SP OptionType [ SP Documentation ] CRLF
         OptionName = Keyword
         OptionType = "Integer" / "TimeInterval" /
"TimeMsecInterval" /
           "DataSize" / "Float" / "Boolean" / "Time" / "CommaList" /
           "Dependent" / "Virtual" / "String" / "LineList"
         Documentation = Text
      Note: The incorrect spelling "Dependant" was used from the time
this key
      was introduced in Tor 0.1.1.4-alpha until it was corrected in
Tor
      0.3.0.2-alpha.  It is recommended that clients accept both
spellings.

    "config/defaults"
      A series of lines listing default values for each configuration
      option. Options which don't have a valid default don't show up
      in the list.  Introduced in Tor 0.2.4.1-alpha.
         OptionName SP OptionValue CRLF
         OptionName = Keyword
         OptionValue = Text
```

```
      arguments as described in section 4.1.

    "stream-status"
      A series of lines as for a stream status event.  Each is of the
form:
         StreamID SP StreamStatus SP CircuitID SP Target CRLF

    "orconn-status"
      A series of lines as for an OR connection status event.  In Tor
      0.1.2.2-alpha with feature VERBOSE_NAMES enabled and in Tor
      0.2.2.1-alpha and later by default, each line is of the form:
         LongName SP ORStatus CRLF

     In Tor versions 0.1.2.2-alpha through 0.2.2.1-alpha with feature
     VERBOSE_NAMES turned off and before version 0.1.2.2-alpha, each
line
     is of the form:
         ServerID SP ORStatus CRLF

    "entry-guards"
      A series of lines listing the currently chosen entry guards, if
any.
      In Tor 0.1.2.2-alpha with feature VERBOSE_NAMES enabled and in
Tor
      0.2.2.1-alpha and later by default, each line is of the form:
         LongName SP Status [SP ISOTime] CRLF

     In Tor versions 0.1.2.2-alpha through 0.2.2.1-alpha with feature
     VERBOSE_NAMES turned off and before version 0.1.2.2-alpha, each
line
     is of the form:
         ServerID2 SP Status [SP ISOTime] CRLF
         ServerID2 = Nickname / 40*HEXDIG

      The definition of Status is the same for both:
         Status = "up" / "never-connected" / "down" /
                  "unusable" / "unlisted"

      [From 0.1.1.4-alpha to 0.1.1.10-alpha, entry-guards was called
       "helper-nodes". Tor still supports calling "helper-nodes", but
it
        is deprecated and should not be used.]

      [Older versions of Tor (before 0.1.2.x-final) generated 'down'
instead
       of unlisted/unusable. Between 0.1.2.x-final and 0.2.6.3-alpha,
       'down' was never generated.]

      [XXXX ServerID2 differs from ServerID in not prefixing
fingerprints
       with a $.  This is an implementation error.  It would be nice
to add
       the $ back in if we can do so without breaking compatibility.]

    "traffic/read" -- Total bytes read (downloaded).

    "version" -- The version of the server's software, which MAY
include the
      name of the software, such as "Tor 0.0.9.4".  The name of the
software,
      if absent, is assumed to be "Tor".

    "config-file" -- The location of Tor's configuration file
("torrc").

    "config-defaults-file" -- The location of Tor's configuration
      defaults file ("torrc.defaults").  This file gets parsed before
      torrc, and is typically used to replace Tor's default
      configuration values. [First implemented in 0.2.3.9-alpha.]

    "config-text" -- The contents that Tor would write if you send it
      a SAVECONF command, so the controller can write the file to
      disk itself. [First implemented in 0.2.2.7-alpha.]

    "exit-policy/default" -- The default exit policy lines that Tor
will
      *append* to the ExitPolicy config option.

    "exit-policy/reject-private/default" -- The default exit policy
lines
      that Tor will *prepend* to the ExitPolicy config option when
      ExitPolicyRejectPrivate is 1.

    "exit-policy/reject-private/relay" -- The relay-specific exit
policy
      lines that Tor will *prepend* to the ExitPolicy config option
based
      on the current values of ExitPolicyRejectPrivate and
      ExitPolicyRejectLocalInterfaces. These lines are based on the
public
      addresses configured in the torrc and present on the relay's
      interfaces. Will send 552 error if the server is not running as
      onion router. Will send 551 on internal error which may be
transient.

    "exit-policy/ipv4"
    "exit-policy/ipv6"
    "exit-policy/full" -- This OR's exit policy, in IPv4-only, IPv6-
only, or
      all-entries flavors. Handles errors in the same way as "exit-
policy/
      reject-private/relay" does.

    "desc/id/<OR identity>" or "desc/name/<OR nickname>" -- the latest
      server descriptor for a given OR.  (Note that modern Tor clients
      do not download server descriptors by default, but download
      microdescriptors instead.  If microdescriptors are enabled,
you'll
      need to use "md" instead.)
```

"md/all" -- all known microdescriptors for the entire Tor network.
Each microdescriptor is terminated by a newline.
[First implemented in 0.3.5.1-alpha]

"md/id/<OR identity>" or "md/name/<OR nickname>" -- the latest
microdescriptor for a given OR. Empty if we have no
microdescriptor for
that OR (because we haven't downloaded one, or it isn't in the
consensus). [First implemented in 0.2.3.8-alpha.]

"desc/download-enabled" -- "1" if we try to download router
descriptors;
"0" otherwise. [First implemented in 0.3.2.1-alpha]

"md/download-enabled" -- "1" if we try to download
microdescriptors;
"0" otherwise. [First implemented in 0.3.2.1-alpha]

"dormant" -- A nonnegative integer: zero if Tor is currently
active and
building circuits, and nonzero if Tor has gone idle due to lack
of use
or some similar reason. [First implemented in 0.2.3.16-alpha]

"desc-annotations/id/<OR identity>" -- outputs the annotations
string
(source, timestamp of arrival, purpose, etc) for the
corresponding
descriptor. [First implemented in 0.2.0.13-alpha.]

"extra-info/digest/<digest>" -- the extrainfo document whose
digest (in
hex) is <digest>. Only available if we're downloading extra-
info
documents.

"ns/id/<OR identity>" or "ns/name/<OR nickname>" -- the latest
router
status info (v3 directory style) for a given OR. Router status
info is as given in dir-spec.txt, and reflects the latest
consensus opinion about the
router in question. Like directory clients, controllers MUST
tolerate unrecognized flags and lines. The published date and
descriptor digest are those believed to be best by this Tor,
not necessarily those for a descriptor that Tor currently has.
[First implemented in 0.1.2.3-alpha.]
[In 0.2.0.9-alpha this switched from v2 directory style to v3]

"ns/all" -- Router status info (v3 directory style) for all ORs we
that the consensus has an opinion about, joined by newlines.
[First implemented in 0.1.2.3-alpha.]
[In 0.2.0.9-alpha this switched from v2 directory style to v3]

"ns/purpose/<purpose>" -- Router status info (v3 directory style)
for all ORs of this purpose. Mostly designed for /ns/purpose/
bridge
queries.
[First implemented in 0.2.0.13-alpha.]
[In 0.2.0.9-alpha this switched from v2 directory style to v3]
[In versions before 0.4.1.1-alpha we set the Running flag on
bridges when /ns/purpose/bridge is accessed]
[In 0.4.1.1-alpha we set the Running flag on bridges when the
bridge networkstatus file is written to disk]

"desc/all-recent" -- the latest server descriptor for every router
that
Tor knows about. (See md note about "desc/id" and "desc/name"
above.)

"network-status" -- [Deprecated in 0.3.1.1-alpha, removed
in 0.4.5.1-alpha.]

"address-mappings/all"
"address-mappings/config"
"address-mappings/cache"
"address-mappings/control" -- a \r\n-separated list of address
mappings, each in the form of "from-address to-address expiry".
The 'config' key returns those address mappings set in the
configuration; the 'cache' key returns the mappings in the
client-side DNS cache; the 'control' key returns the mappings
set
via the control interface; the 'all' target returns the mappings
set through any mechanism.
Expiry is formatted as with ADDRMAP events, except that "expiry"
is
always a time in UTC or the string "NEVER"; see section 4.1.7.
First introduced in 0.2.0.3-alpha.

"addr-mappings/*" -- as for address-mappings/*, but without the
expiry portion of the value. Use of this value is deprecated
since 0.2.0.3-alpha; use address-mappings instead.

"address" -- the best guess at our external IP address. If we
have no guess, return a 551 error. (Added in 0.1.2.2-alpha)

"address/v4"
"address/v6"
the best guess at our respective external IPv4 or IPv6 address.
If we have no guess, return a 551 error. (Added in 0.4.5.1-
alpha)

"fingerprint" -- the contents of the fingerprint file that Tor
writes as a relay, or a 551 if we're not a relay currently.
(Added in 0.1.2.3-alpha)

"circuit-status"
A series of lines as for a circuit status event. Each line is of
the form described in section 4.1.1, omitting the initial
"650 CIRC ". Note that clients must be ready to accept
additional