

---

# pynghttp2 Documentation

*Release 0.1.0*

**Lucas Kahlert**

**Jun 01, 2023**



# CONTENTS

<b>1</b>	<b>Quickstart</b>	<b>3</b>
<b>2</b>	<b>API</b>	<b>5</b>
2.1	High-level API . . . . .	5
2.2	Advanced server and client sessions . . . . .	5
2.3	HTTP/2 Requests and Responses . . . . .	7
2.4	Flow Control with StreamReaders . . . . .	8
2.5	Ctypes Bindings . . . . .	9
<b>3</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>
	<b>Index</b>	<b>37</b>



pynghttp2 are simple asyncio Python bindings based on ctypes for the [nghttp2](#) library. The only thing you need is a [libnghttp2](#) version on your system.

The project was created in the context of a student work for an HTTP/2 protocol gateway in the [μPCN](#) project - an implementation of Delay-tolerant Networking (DTN) protocols.



**QUICKSTART**





## 2.1 High-level API

High-Level functions for simple requests

```
from pynghttp2 import http2

resp = await http2.get("http://localhost:8000/README.rst")
content = await resp.text()
```

```
coroutine pynghttp2.http2.get(*args, **kwargs)
```

```
coroutine pynghttp2.http2.post(*args, **kwargs)
```

```
coroutine pynghttp2.http2.request(method, url, headers=None, data=None, host=None, port=None,
                                   loop=None)
```

## 2.2 Advanced server and client sessions

Asyncio HTTP/2 client and server sessions based on the [nghttp2](#) Python wrapper around the nghttp2 API.

```
class pynghttp2.sessions.BaseHTTP2(settings, loop)
```

Bases: Protocol

```
can_write_stream(stream_id)
```

```
connection_lost(exc)
```

Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

```
connection_made(transport)
```

Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for `data_received()` calls. When the connection is closed, `connection_lost()` is called.

```
content_sent(stream_id)
```

**data\_received**(*data*)

Called when some data is received.

The argument is a bytes object.

**coroutine drain**()

**flush**()

**goaway\_received**(*error\_code*)

**goaway\_sent**(*error\_code*)

**headers\_sent**(*stream\_id*)

**on\_data\_chunk\_recv**(*stream\_id*, *chunk*)

**on\_header**(*stream\_id*, *header*)

**pause\_writing**()

Called when the transport's buffer goes over the high-water mark.

Pause and resume calls are paired – `pause_writing()` is called once when the buffer goes strictly over the high-water mark (even if subsequent writes increases the buffer size even more), and eventually `resume_writing()` is called once when the buffer size reaches the low-water mark.

Note that if the buffer size equals the high-water mark, `pause_writing()` is not called – it must go strictly over. Conversely, `resume_writing()` is called when the buffer size is equal or lower than the low-water mark. These end conditions are important to ensure that things go as expected when either mark is zero.

NOTE: This is the only Protocol callback that is not called through `EventLoop.call_soon()` – if it were, it would have no effect when it's most needed (when the app keeps writing without yielding until `pause_writing()` is called).

**resume\_writing**()

Called when the transport's buffer drains below the low-water mark.

See `pause_writing()` for details.

**settings\_updated**()

**stream\_closed**(*stream\_id*, *error\_code*)

**submit\_response**(*stream\_id*, *resp*)

**coroutine terminate**(*error\_code*=*error\_code.NO\_ERROR*)

**coroutine wait\_for\_window\_update**(*stream\_id*)

**window\_update\_received**(*stream\_id*)

**class** `pynghttp2.sessions.ClientProtocol`(*settings*, *loop*)

Bases: [`BaseHTTP2`](#)

**begin\_headers**(*stream\_id*)

**content\_received**(*stream\_id*)

**establish\_session**()

**headers\_received**(*stream\_id*)

```

    stream_closed(stream_id, error_code)

    submit_request(req, resp)

class pynghttp2.sessions.ClientSession(host, port, settings=None, loop=None)
    Bases: object
    get(url, headers=None)
    post(url, headers=None, data=None)
    request(method=None, url=None, headers=None, data=None)
    request_allowed()
    coroutine start()
    coroutine terminate(error_code=error_code.NO_ERROR)

class pynghttp2.sessions.ServerProtocol(on_request_callback, settings, loop)
    Bases: BaseHTTP2
    begin_headers(stream_id)
    content_received(stream_id)
    establish_session()
    headers_received(stream_id)

class pynghttp2.sessions.ServerSession(host, port, settings=None, loop=None)
    Bases: object
    close()
    coroutine start()
    coroutine wait_closed()

```

## 2.3 HTTP/2 Requests and Responses

```

class pynghttp2.messages.Direction(value)
    Bases: IntEnum
    An enumeration.
    RECEIVING = 0
    SENDING = 1

class pynghttp2.messages.HTTP2Message(protocol, stream_id, direction, headers=None, data=None,
                                         loop=None)
    Bases: object
    property authority
    property content_length

```

```
content_sent()
property content_type
headers_received()
headers_sent()
coroutine json()
property method
property path
coroutine read()
property scheme
set_exception(exc)
stream_closed(error_code)
property stream_id
coroutine text()

class pynghttp2.messages.Request(protocol, stream_id, direction, headers=None, data=None, loop=None)
    Bases: HTTP2Message
    property method
    property path
    response(status, data=None, headers=None)

class pynghttp2.messages.Response(protocol, stream_id, direction, headers=None, data=None, loop=None)
    Bases: HTTP2Message
    property status

exception pynghttp2.messages.StreamClosedError(stream_id=None,
                                                error_code=error_code.NO_ERROR)
    Bases: ConnectionError
```

## 2.4 Flow Control with StreamReaders

```
pynghttp2.streams.DATA_MIN_SIZE = 128
    Minimal size of a DATA block. If a stream reader

class pynghttp2.streams.StreamReader(loop=None)
    Bases: object
    at_eof()
        Return True if the buffer is empty and feed_eof() was called.
    coroutine drain()
```

```

empty()

feed_data(data)

feed_eof()

is_deferred()

is_eof()
    Return True if feed_eof() was called.

coroutine read(n=-1)

read_nowait(n)

coroutine readany()

reading_deferred()
    Deferr reading until feed_data is called

reading_resumed()

set_exception(exc)

set_protocol(protocol, stream_id)

coroutine wait_eof()

```

## 2.5 Ctypes Bindings

Python wrapper for the nghttp2 API

The nghttp2 API is already object orientated but this module wraps the low-level OOP in Python's own object model.

```

class pynghttp2.nghttp2.Options(builtin_rcv_extension_type=None,
                                max_deflate_dynamic_table_size=None,
                                max_reserved_remote_streams=None,
                                max_send_header_block_length=None, no_auto_ping_ack=None,
                                no_auto_window_update=None, no_closed_streams=None,
                                no_http_messaging=None, no_rcv_client_magic=None,
                                peer_max_concurrent_streams=None, user_rcv_extension_type=None)

```

Bases: `object`

**set\_builtin\_rcv\_extension\_type(*type*)**

Sets extension frame type the application is willing to receive using builtin handler. The type is the extension frame type to receive, and must be strictly greater than 0x9. Otherwise, this function does nothing. The application can call this function multiple times to set more than one frame type to receive. The application does not have to call this function if it just sends extension frames.

If same frame type is passed to both `nghttp2_option_set_builtin_rcv_extension_type()` and `nghttp2_option_set_user_rcv_extension_type()`, the latter takes precedence.

**set\_max\_deflate\_dynamic\_table\_size(*val*)**

This option sets the maximum dynamic table size for deflating header fields. The default value is 4KiB. In HTTP/2, receiver of deflated header block can specify maximum dynamic table size. The actual maximum size is the minimum of the size receiver specified and this option value.

**Parameters**

**val** (*int*) – Maximum dynamic table size

**set\_max\_reserved\_remote\_streams(val)**

RFC 7540 does not enforce any limit on the number of incoming reserved streams (in RFC 7540 terms, streams in reserved (remote) state). This only affects client side, since only server can push streams. Malicious server can push arbitrary number of streams, and make client's memory exhausted. This option can set the maximum number of such incoming streams to avoid possible memory exhaustion. If this option is set, and pushed streams are automatically closed on reception, without calling user provided callback, if they exceed the given limit. The default value is 200. If session is configured as server side, this option has no effect. Server can control the number of streams to push.

**Parameters**

**val** (*int*) – Max. number of reserved streams

**set\_max\_send\_header\_block\_length(val)**

This option sets the maximum length of header block (a set of header fields per one HEADERS frame) to send. The length of a given set of header fields is calculated using `nghttp2_hd_deflate_bound()`. The default value is 64KiB. If application attempts to send header fields larger than this limit, the transmission of the frame fails with error code `error.FRAME_SIZE_ERROR`.

**Parameters**

**val** (*int*) – Max. length of header block to send

**set\_no\_auto\_ping\_ack(val)**

This option prevents the library from sending PING frame with ACK flag set automatically when PING frame without ACK flag set is received. If this option is set to True, the library won't send PING frame with ACK flag set in the response for incoming PING frame. The application can send PING frame with ACK flag set using `nghttp2_submit_ping()` with `typedefs.flag.ACK` as flags parameter.

**Parameters**

**val** (*bool*) – If True, the library won't answer PING frames automatically

**set\_no\_auto\_window\_update(val)**

This option prevents the library from sending WINDOW\_UPDATE for a connection automatically. If this option is set to True, the library won't send WINDOW\_UPDATE for DATA until application calls `nghttp2_session_consume()` to indicate the consumed amount of data. Don't use `nghttp2_submit_window_update()` for this purpose. By default, this option is set to zero.

**Parameters**

**val** (*bool*) – If True, disables automatic WINDOW\_UPDATE frames

**set\_no\_closed\_streams(val)**

This option prevents the library from retaining closed streams to maintain the priority tree. If this option is set to True, applications can discard closed stream completely to save memory.

**Parameters**

**val** (*bool*) – If True, library will not retain closed streams

**set\_no\_http\_messaging(val)**

By default, nghttp2 library enforces subset of HTTP Messaging rules described in HTTP/2 specification, section 8. See HTTP Messaging section for details. For those applications who use nghttp2 library as non-HTTP use, give True to val to disable this enforcement. Please note that disabling this feature does not change the fundamental client and server model of HTTP. That is, even if the validation is disabled, only client can send requests.

**Parameters**

**val** (*bool*) – If True, disables HTTP Messaging rules

**set\_no\_recv\_client\_magic(val)**

By default, nghttp2 library, if configured as server, requires first 24 bytes of client magic byte string (MAGIC). In most cases, this will simplify the implementation of server. But sometimes server may want to detect the application protocol based on first few bytes on clear text communication.

If this option is used with True val, nghttp2 library does not handle MAGIC. It still checks following SETTINGS frame. This means that applications should deal with MAGIC by themselves.

If this option is not used or used with zero value, if MAGIC does not match NGHTTP2\_CLIENT\_MAGIC, nghttp2\_session\_recv() and nghttp2\_session\_mem\_recv() will return error error.BAD\_CLIENT\_MAGIC, which is fatal error.

**Parameters**

**val** (*bool*) – If True, library does not handle MAGIC

**set\_peer\_max\_concurrent\_streams(val)**

This option sets the attr:settings.MAX\_CONCURRENT\_STREAMS value of remote endpoint as if it is received in SETTINGS frame. Without specifying this option, before the local endpoint receives attr:settings.MAX\_CONCURRENT\_STREAMS in SETTINGS frame from remote endpoint, attr:settings.MAX\_CONCURRENT\_STREAMS is unlimited. This may cause problem if local endpoint submits lots of requests initially and sending them at once to the remote peer may lead to the rejection of some requests. Specifying this option to the sensible value, say 100, may avoid this kind of issue. This value will be overwritten if the local endpoint receives attr:settings.MAX\_CONCURRENT\_STREAMS from the remote endpoint.

**Parameters**

**val** (*int*) – Max. number of concurrent streams per peer before local SETTINGS frame is send

**set\_user\_recv\_extension\_type(val)**

Sets extension frame type the application is willing to handle with user defined callbacks (see nghttp2\_on\_extension\_chunk\_recv\_callback and nghttp2\_unpack\_extension\_callback). The type is extension frame type, and must be strictly greater than 0x9. Otherwise, this function does nothing. The application can call this function multiple times to set more than one frame type to receive. The application does not have to call this function if it just sends extension frames.

**Parameters**

**val** (*int*) – type the application is willing to handle with user defined callbacks

**class** pynghttp2.nghttp2.Session(*type, callbacks, user\_data=None, options=None*)

Bases: `object`

**consume**(*stream\_id, size*)

**consume\_connection**(*size*)

Like `consume()`, but this only tells library that size bytes were consumed only for connection level. Note that HTTP/2 maintains connection and stream level flow control windows independently.

**Parameters**

**size** (*int*) – Bytes consumed for the connection window

**Raises**

- **ValueError** – If automatic WINDOW\_UPDATE is enabled
- **MemoryError** – Out of memory

**consume\_stream**(*stream\_id, size*)

Like `consume()`, but this only tells library that size bytes were consumed only for stream denoted by stream\_id. Note that HTTP/2 maintains connection and stream level flow control windows independently.

**Parameters**

- **stream\_id** (*int*) – Stream ID for which the window is maintained
- **size** (*int*) – Bytes consumed for the stream window

**Raises**

- **ValueError** – If automatic WINDOW\_UPDATE is enabled or the stream ID is 0
- **MemoryError** – Out of memory

**get\_local\_settings(settings\_id)**

Returns the value of SETTINGS id of local endpoint acknowledged by the remote endpoint. The id must be one of the values defined in `typedes.nghttp2_settings_id`.

**Parameters**

**settings\_id** (`.typedefs.settings_id`) – Setting that should be returned

**Returns**

Current settings value for the local endpoint

**Return type**

`int`

**get\_local\_window\_size()****get\_remote\_settings(settings\_id)**

Returns the value of SETTINGS id notified by a remote endpoint. The id must be one of values defined in `typedefs.settings_id`.

**Parameters**

**settings\_id** (`.typedefs.settings_id`) – Setting that should be returned

**Returns**

Current settings value for the remote endpoint

**Return type**

`int`

**get\_remote\_window\_size()****get\_stream\_local\_close(stream\_id)**

Returns True if local peer half closed the given stream `stream_id`. Returns False if it did not.

**Parameters**

**stream\_id** (*int*) – Stream identifier

**Returns**

True if the local peer has closed the stream

**Return type**

`bool`

**Raises**

**ValueError** – if no such stream exists.

**get\_stream\_local\_window\_size(stream\_id)****get\_stream\_remote\_close(stream\_id)**

Returns True if remote peer half closed the given stream `stream_id`. Returns False if it did not.

**Parameters**

**stream\_id** (*int*) – Stream identifier



**Returns**

True if the remote peer has closed the stream

**Return type**

`bool`

**Raises**

**ValueError** – if no such stream exists.

`get_stream_remote_window_size(stream_id)`

`get_stream_user_data(stream_id)`

`is_open()`

`mem_rcv(data)`

`mem_send()`

`request_allowed()`

Returns nonzero if new request can be sent from local endpoint.

This function return False if request is not allowed for this session. There are several reasons why request is not allowed. Some of the reasons are:

- session is server
- stream ID has been spent
- GOAWAY has been sent or received

The application can call `submit_request()` without consulting this function. In that case, `submit_request()` may raises an error. Or, request is failed to sent, and `typedefs.on_stream_close_callback` is called.

**Returns**

True if a request can be sent

**Return type**

`bool`

`resume_data(stream_id)`

`send()`

`set_stream_user_data(stream_id, user_data)`

`stream_exists(stream_id)`

`submit_data(stream_id, provider, flags=flag.END_STREAM)`

`submit_headers(flags, stream_id, headers, priority=None)`

`submit_request(headers, provider=None, stream_data=None, priority=None)`

`submit_response(stream_id, headers, provider=None)`

`submit_settings(settings)`

`terminate(error_code=error_code.NO_ERROR)`

`want_read()`

**want\_write()**

`pynghttp2.nghttp2.cast_py_object(ptr)`

`pynghttp2.nghttp2.session_get_stream_user_data(session, stream_id)`

`pynghttp2.nghttp2.session_set_stream_user_data(session, stream_id, user_data)`

**class** `pynghttp2.nghttp2.session_type(value)`

Bases: `IntEnum`

An enumeration.

**CLIENT** = 0

**SERVER** = 1

`pynghttp2.nghttp2.version()`

Type definitions compatible with the libnghttp2 API

**class** `pynghttp2.typedefs.data`

Bases: `Structure`

The DATA frame. The received data is delivered via `on_data_chunk_rcv_callback`.

**padlen**

The length of the padding in this frame. This includes `PAD_HIGH` and `PAD_LOW`.

**Type**

`size_t`

**padlen**

Structure/Union member

**class** `pynghttp2.typedefs.data_flag(value)`

Bases: `IntFlag`

The flags used to set in `data_flags` output parameter in `data_source_read_callback`.

**EOF** = 1

Indicates EOF was sensed.

**NONE** = 0

No flag set.

**NO\_COPY** = 4

Indicates that application will send complete DATA frame in `send_data_callback`.

**NO\_END\_STREAM** = 2

Indicates that `END_STREAM` flag must not be set even if EOF is set. Usually this flag is used to send trailer fields with `submit_request()` or `submit_response()`.

**class** `pynghttp2.typedefs.data_provider`

Bases: `Structure`

This struct represents the data source and the way to read a chunk of data from it.

**source**

The data source.

**Type**

*data\_source*

**read\_callback**

The callback function to read a chunk of data from the source.

**Type**

*data\_source\_read\_callback*

**read\_callback**

Structure/Union member

**source**

Structure/Union member

**class** pynghttp2.typedefs.**data\_source**

Bases: Union

This union represents the some kind of data source passed to *data\_source\_read\_callback*.

**fd**

The integer field, suitable for a file descriptor.

**Type**

*int*

**ptr**

The pointer to an arbitrary object.

**Type**

*\*void*

**fd**

Structure/Union member

**ptr**

Structure/Union member

**pynghttp2.typedefs.data\_source\_read\_callback**

alias of CFunctionType

**class** pynghttp2.typedefs.**error**(*value*)

Bases: *IntEnum*

Error codes used in the nghttp2 library. The code range is [-999, -500], inclusive.

**BAD\_CLIENT\_MAGIC = -903**

Invalid client magic (see NGHTTP2\_CLIENT\_MAGIC) was received and further processing is not possible.

**BUFFER\_ERROR = -502**

Out of buffer space.

**CALLBACK\_FAILURE = -902**

The user callback function failed. This is a fatal error.

**CANCEL = -535**

Indicates that a processing was canceled.

**DATA\_EXIST = -529**

DATA or HEADERS frame for a given stream has been already submitted and has not been fully processed yet. Application should wait for the transmission of the previously submitted frame before submitting another.

**DEFERRED = -508**

Used as a return value from `data_source_read_callback()` to indicate that data transfer is postponed. See `data_source_read_callback()` for details.

**DEFERRED\_DATA\_EXIST = -515**

Another DATA frame has already been deferred.

**EOF = -507**

The peer performed a shutdown on the connection.

**FATAL = -900**

The errors < FATAL mean that the library is under unexpected condition and processing was terminated (e.g., out of memory). If application receives this error code, it must stop using that session object and only allowed operation for that object is deallocate it using `session_del()`.

**FLOODED = -904**

Possible flooding by peer was detected in this HTTP/2 session. Flooding is measured by how many PING and SETTINGS frames with ACK flag set are queued for transmission. These frames are response for the peer initiated frames, and peer can cause memory exhaustion on server side to send these frames forever and does not read network.

**FLOW\_CONTROL = -524**

Flow control error

**FRAME\_SIZE\_ERROR = -522**

The length of the frame is invalid, either too large or too small.

**GOAWAY\_ALREADY\_SENT = -517**

GOAWAY has already been sent.

**HEADER\_COMP = -523**

Header block inflate/deflate error.

**HTTP\_HEADER = -531**

Invalid HTTP header field was received and stream is going to be closed.

**HTTP\_MESSAGING = -532**

Violation in HTTP messaging rule.

**INSUFF\_BUFSIZE = -525**

Insufficient buffer size given to function.

**INTERNAL = -534**

Unexpected internal error, but recovered.

**INVALID\_ARGUMENT = -501**

Invalid argument passed.

**INVALID\_FRAME = -506**

The frame is invalid.

**INVALID\_HEADER\_BLOCK = -518**

The received frame contains the invalid header block (e.g., There are duplicate header names; or the header names are not encoded in US-ASCII character set and not lower cased; or the header name is zero-length string; or the header value contains multiple in-sequence NUL bytes).

**INVALID\_STATE = -519**

Indicates that the context is not suitable to perform the requested operation.

**INVALID\_STREAM\_ID = -513**

The stream ID is invalid.

**INVALID\_STREAM\_STATE = -514**

The state of the stream is not valid (e.g., DATA cannot be sent to the stream if response HEADERS has not been sent).

**NOMEM = -901**

Out of memory. This is a fatal error.

**PAUSE = -526**

Callback was paused by the application

**PROTO = -505**

General protocol error

**PUSH\_DISABLED = -528**

The server push is disabled.

**REFUSED\_STREAM = -533**

Stream was refused.

**SESSION\_CLOSING = -530**

The current session is closing due to a connection error or session\_terminate\_session() is called.

**SETTINGS\_EXPECTED = -536**

When a local endpoint expects to receive SETTINGS frame, it receives an other type of frame.

**START\_STREAM\_NOT\_ALLOWED = -516**

Starting new stream is not allowed (e.g., GOAWAY has been sent and/or received).

**STREAM\_CLOSED = -510**

The stream is already closed; or the stream ID is invalid.

**STREAM\_CLOSING = -511**

RST\_STREAM has been added to the outbound queue. The stream is in closing state.

**STREAM\_ID\_NOT\_AVAILABLE = -509**

Stream ID has reached the maximum value. Therefore no stream ID is available.

**STREAM\_SHUT\_WR = -512**

The transmission is not allowed for this stream (e.g., a frame with END\_STREAM flag set has already sent).

**TEMPORAL\_CALLBACK\_FAILURE = -521**

The user callback function failed due to the temporal error.

**TOO\_MANY\_INFLIGHT\_SETTINGS = -527**

There are too many in-flight SETTING frame and no more transmission of SETTINGS is allowed.

**UNSUPPORTED\_VERSION = -503**

The specified protocol version is not supported.

**WOULDBLOCK = -504**

Used as a return value from `send_callback`, `recv_callback` and `send_data_callback` to indicate that the operation would block.

**class** pynghttp2.typedefs.**error\_code**(*value*)

Bases: `IntEnum`

The status codes for the RST\_STREAM and GOAWAY frames.

**CANCEL = 8**

CANCEL

**COMPRESSION\_ERROR = 9**

COMPRESSION\_ERROR

**CONNECT\_ERROR = 10**

CONNECT\_ERROR

**ENHANCE\_YOUR\_CALM = 11**

ENHANCE\_YOUR\_CALM

**FLOW\_CONTROL\_ERROR = 3**

FLOW\_CONTROL\_ERROR

**FRAME\_SIZE\_ERROR = 6**

FRAME\_SIZE\_ERROR

**HTTP\_1\_1\_REQUIRED = 13**

HTTP\_1\_1\_REQUIRED

**INADEQUATE\_SECURITY = 12**

INADEQUATE\_SECURITY

**INTERNAL\_ERROR = 2**

INTERNAL\_ERROR

**NO\_ERROR = 0**

No errors.

**PROTOCOL\_ERROR = 1**

PROTOCOL\_ERROR

**REFUSED\_STREAM = 7**

REFUSED\_STREAM

**SETTINGS\_TIMEOUT = 4**

SETTINGS\_TIMEOUT

**STREAM\_CLOSED = 5**

STREAM\_CLOSED

**class** pynghttp2.typedefs.**extension**

Bases: `Structure`

The extension frame. It has following members:

**hd**

The frame header.

**Type***frame\_hd***payload**

The pointer to extension payload. The exact pointer type is determined by hd.type.

Currently, no extension is supported. This is a place holder for the future extensions.

**Type***\*void***hd**

Structure/Union member

**payload**

Structure/Union member

**class** pynghttp2.typedefs.**flag**(*value*)Bases: *IntEnum*

The flags for HTTP/2 frames. This enum defines all flags for all frames.

**ACK = 1**

The ACK flag.

**END\_HEADERS = 4**

The END\_HEADERS flag.

**END\_STREAM = 1**

The END\_STREAM flag.

**NONE = 0**

No flag set.

**PADDED = 8**

The PADDED flag.

**PRIORITY = 32**

The PRIORITY flag.

**class** pynghttp2.typedefs.**frame**Bases: *Union*

This union includes all frames to pass them to various function calls as frame type. The CONTINUATION frame is omitted from here because the library deals with it internally.

**hd**

The frame header, which is convenient to inspect frame header.

**Type***frame\_hd***data**

The DATA frame.

**Type***data*

**headers**

The HEADERS frame.

**Type**

*headers*

**priority**

The PRIORITY frame.

**Type**

*priority*

**rst\_stream**

The RST\_STREAM frame.

**Type**

*rst\_stream*

**settings**

The SETTINGS frame.

**Type**

*settings*

**push\_promise**

The PUSH\_PROMISE frame.

**Type**

*push\_promise*

**ping**

The PING frame.

**Type**

*ping*

**goaway**

The GOAWAY frame.

**Type**

*goaway*

**window\_update**

The WINDOW\_UPDATE frame.

**Type**

*window\_update*

**ext**

The extension frame.

**Type**

*extension*

**data**

Structure/Union member

**ext**

Structure/Union member



**goaway**

Structure/Union member

**hd**

Structure/Union member

**headers**

Structure/Union member

**ping**

Structure/Union member

**priority**

Structure/Union member

**push\_promise**

Structure/Union member

**rst\_stream**

Structure/Union member

**settings**

Structure/Union member

**window\_update**

Structure/Union member

**class** pynghttp2.typedefs.**frame\_hd**

Bases: Structure

The frame header

**length**

The length field of this frame, excluding frame header.

**Type**

size\_t

**stream\_id**

The stream identifier (aka, stream ID)

**Type**

int32\_t

**type**

The type of this frame. See frame\_type().

**Type**

uint8\_t

**flags**

The flags.

**Type**

uint8\_t

**reserved**

Reserved bit in frame header. Currently, this is always set to 0 and application should not expect something useful in here.

**Type**  
uint8\_t

**flags**  
Structure/Union member

**length**  
Structure/Union member

**reserved**  
Structure/Union member

**stream\_id**  
Structure/Union member

**type**  
Structure/Union member

**class** pynghttp2.typedefs.**frame\_type**(*value*)  
Bases: `IntEnum`  
The frame types in HTTP/2 specification.

**ALTSVC = 10**  
The ALTSVC frame, which is defined in RFC 7383.

**CONTINUATION = 9**  
The CONTINUATION frame. This frame type won't be passed to any callbacks because the library processes this frame type and its preceding HEADERS/PUSH\_PROMISE as a single frame.

**DATA = 0**  
The DATA frame.

**GOAWAY = 7**  
The GOAWAY frame.

**HEADERS = 1**  
The HEADERS frame.

**PING = 6**  
The PING frame.

**PRIORITY = 2**  
The PRIORITY frame.

**PUSH\_PROMISE = 5**  
The PUSH\_PROMISE frame.

**RST\_STREAM = 3**  
The RST\_STREAM frame.

**SETTINGS = 4**  
The SETTINGS frame.

**WINDOW\_UPDATE = 8**  
The WINDOW\_UPDATE frame.

**class** pynghttp2.typedefs.goaway

Bases: Structure

The GOAWAY frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**last\_stream\_id**

The last stream stream ID.

**Type**

int32\_t

**error\_code**

The error code. See error\_code.

**Type**

uint32\_t

**\\*opaque\_data**

The additional debug data

**Type**

uint8\_t

**opaque\_data\_len**

The length of opaque\_data member.

**Type**

size\_t

**reserved**

Reserved bit. Currently this is always set to 0 and application should not expect something useful in here.

**Type**

uint8\_t

**error\_code**

Structure/Union member

**hd**

Structure/Union member

**last\_stream\_id**

Structure/Union member

**opaque\_data**

Structure/Union member

**opaque\_data\_len**

Structure/Union member

**reserved**

Structure/Union member

**class** pynghttp2.typedefs.**hd\_inflate\_flag**(*value*)

Bases: [IntFlag](#)

The flags for header inflation.

**EMIT** = 2

Indicates a header was emitted.

**FINAL** = 1

Indicates all headers were inflated.

**NONE** = 0

No flag set.

**class** pynghttp2.typedefs.**headers**

Bases: [Structure](#)

The HEADERS frame. It has the following members:

**hd**

The frame header.

**Type**

[frame\\_hd](#)

**padlen**

The length of the padding in this frame. This includes PAD\_HIGH and PAD\_LOW.

**Type**

[size\\_t](#)

**pri\_spec**

The priority specification

**Type**

[priority\\_spec](#)

**nva**

The name/value pairs.

**Type**

[\\*nv](#)

**nvlen**

The number of name/value pairs in nva.

**Type**

[size\\_t](#)

**cat**

The category of this HEADERS frame.

**Type**

[headers\\_category](#)

**cat**

Structure/Union member

**hd**

Structure/Union member

**nva**

Structure/Union member

**nvlen**

Structure/Union member

**padlen**

Structure/Union member

**pri\_spec**

Structure/Union member

**class** pynghttp2.typedefs.headers\_category(*value*)Bases: `IntEnum`

The category of HEADERS, which indicates the role of the frame. In HTTP/2 spec, request, response, push response and other arbitrary headers (e.g., trailer fields) are all called just HEADERS. To give the application the role of incoming HEADERS frame, we define several categories.

**HEADERS = 3**

The HEADERS frame which does not apply for the above categories, which is analogous to HEADERS in SPDY. If non-final response (e.g., status 1xx) is used, final response HEADERS frame will be categorized here.

**PUSH\_RESPONSE = 2**

The HEADERS frame is the first headers sent against reserved stream.

**REQUEST = 0**

The HEADERS frame is opening new stream, which is analogous to SYN\_STREAM in SPDY.

**RESPONSE = 1**

The HEADERS frame is the first response headers, which is analogous to SYN\_REPLY in SPDY.

**class** pynghttp2.typedefs.infoBases: `Structure`**age**

Structure/Union member

**property proto****proto\_str**

Structure/Union member

**property version****property version\_info****version\_num**

Structure/Union member

**version\_str**

Structure/Union member

**class** pynghttp2.typedefs.nvBases: `Structure`

The name/value pair, which mainly used to represent header fields.

**name (uint8\_t\*):**

The name byte string. If this struct is presented from library (e.g., `on_frame_rcv_callback`), name is guaranteed to be NULL-terminated. For some callbacks (`before_frame_send_callback`, `on_frame_send_callback`, and `on_frame_not_send_callback`), it may not be NULL-terminated if header field is passed from application with the flag `NO_COPY_NAME`). When application is constructing this struct, name is not required to be NULL-terminated.

**value (uint8\_t\*):**

The value byte string. If this struct is presented from library (e.g., `on_frame_rcv_callback`), value is guaranteed to be NULL-terminated. For some callbacks (`before_frame_send_callback`, `on_frame_send_callback`, and `on_frame_not_send_callback`), it may not be NULL-terminated if header field is passed from application with the flag `NO_COPY_VALUE`). When application is constructing this struct, value is not required to be NULL-terminated.

**namelen (size\_t):**

The length of the name, excluding terminating NULL.

**valuelen (size\_t):**

The length of the value, excluding terminating NULL.

**flags (uint8\_t):**

Bitwise OR of one or more of [nv\\_flag](#).

**flags**

Structure/Union member

**name**

Structure/Union member

**namelen**

Structure/Union member

**value**

Structure/Union member

**valuelen**

Structure/Union member

**class** `pynghttp2.typedefs.nv_flag(value)`

Bases: [IntFlag](#)

The flags for header field name/value pair.

**NONE = 0**

No flag set.

**NO\_COPY\_NAME = 2**

This flag is set solely by application. If this flag is set, the library does not make a copy of header field name. This could improve performance.

**NO\_COPY\_VALUE = 4**

This flag is set solely by application. If this flag is set, the library does not make a copy of header field value. This could improve performance.

**NO\_INDEX = 1**

Indicates that this name/value pair must not be indexed (“Literal Header Field never Indexed” representation must be used in HPACK encoding). Other implementation calls this bit as “sensitive”.

`pynghttp2.typedefs.on_begin_headers_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.on_data_chunk_rcv_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.on_frame_rcv_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.on_frame_send_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.on_header_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.on_stream_close_callback`  
 alias of `CFunctionType`

`pynghttp2.typedefs.option_p`  
 alias of `c_void_p`

**class** `pynghttp2.typedefs.ping`

Bases: `Structure`

The PING frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**opaque\_data**

The opaque data

**Type**

`uint8_t[8]`

**hd**

Structure/Union member

**opaque\_data**

Structure/Union member

**class** `pynghttp2.typedefs.priority`

Bases: `Structure`

The PRIORITY frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**pri\_spec**

The priority specification.

**Type**

*priority\_spec*

**hd**

Structure/Union member

**pri\_spec**

Structure/Union member

**class** pynghttp2.typedefs.**priority\_spec**

Bases: Structure

The structure to specify stream dependency

**stream\_id**

The stream ID of the stream to depend on. Specifying 0 makes stream not depend any other stream.

**Type**

int32\_t

**weight**

The weight of this dependency.

**Type**

int32\_t

**exclusive**

nonzero means exclusive dependency

**Type**

uint8\_t

**exclusive**

Structure/Union member

**stream\_id**

Structure/Union member

**weight**

Structure/Union member

**class** pynghttp2.typedefs.**push\_promise**

Bases: Structure

The PUSH\_PROMISE frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**padlen**

The length of the padding in this frame. This includes PAD\_HIGH and PAD\_LOW.

**Type**

size\_t

**nva**

The name/value pairs.

**Type**

*\*nv*



**nvlen**

The number of name/value pairs in nva.

**Type**

size\_t

**promised\_stream\_id**

The promised stream ID

**Type**

int32\_t

**reserved**

Reserved bit. Currently this is always set to 0 and application should not expect something useful in here.

**Type**

uint8\_t

**hd**

Structure/Union member

**nva**

Structure/Union member

**nvlen**

Structure/Union member

**padlen**

Structure/Union member

**promised\_stream\_id**

Structure/Union member

**reserved**

Structure/Union member

**class** pynghttp2.typedefs.rst\_stream

Bases: Structure

The RST\_STREAM frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**error\_code**

The error code. See error\_code.

**Type**

uint32\_t

**error\_code**

Structure/Union member

**hd**

Structure/Union member

pynghttp2.typedefs.**send\_callback**

alias of CFunctionType

pynghttp2.typedefs.**send\_data\_callback**

alias of CFunctionType

pynghttp2.typedefs.**session\_callbacks\_p**

alias of `c_void_p`

pynghttp2.typedefs.**session\_p**

alias of `c_void_p`

**class** pynghttp2.typedefs.**settings**

Bases: Structure

The SETTINGS frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**niv**

The number of SETTINGS ID/Value pairs in iv.

**Type**

size\_t

**iv**

The pointer to the array of SETTINGS ID/Value pair.

**Type**

\*settings\_entry

**hd**

Structure/Union member

**iv**

Structure/Union member

**niv**

Structure/Union member

**class** pynghttp2.typedefs.**settings\_entry**

Bases: Structure

The SETTINGS ID/Value pair. It has the following members:

**settings\_id**

The SETTINGS ID. See settings\_id.

**Type**

int32\_t

**value**

The value of this entry.

**Type**

uint32\_t

**settings\_id**

Structure/Union member

**value**

Structure/Union member

**class** pynghttp2.typedefs.**settings\_id**(*value*)Bases: `IntEnum`

The SETTINGS ID

**ENABLE\_PUSH** = 2

SETTINGS\_ENABLE\_PUSH

**HEADER\_TABLE\_SIZE** = 1

SETTINGS\_HEADER\_TABLE\_SIZE

**INITIAL\_WINDOW\_SIZE** = 4

SETTINGS\_INITIAL\_WINDOW\_SIZE

**MAX\_CONCURRENT\_STREAMS** = 3

SETTINGS\_MAX\_CONCURRENT\_STREAMS

**MAX\_FRAME\_SIZE** = 5

SETTINGS\_MAX\_FRAME\_SIZE

**MAX\_HEADER\_LIST\_SIZE** = 6

SETTINGS\_MAX\_HEADER\_LIST\_SIZE

pynghttp2.typedefs.**stream\_p**alias of `c_void_p`**class** pynghttp2.typedefs.**stream\_proto\_state**(*value*)Bases: `IntEnum`

State of stream as described in RFC 7540.

**CLOSED** = 7

closed state.

**HALF\_CLOSED\_LOCAL** = 5

half closed (local) state.

**HALF\_CLOSED\_REMOTE** = 6

half closed (remote) state.

**IDLE** = 1

idle state.

**OPEN** = 2

open state.

**RESERVED\_LOCAL** = 3

reserved (local) state.

**RESERVED\_REMOTE** = 4

reserved (remote) state.

**class** pynghttp2.typedefs.**window\_update**

Bases: Structure

The WINDOW\_UPDATE frame. It has the following members:

**hd**

The frame header.

**Type**

*frame\_hd*

**window\_size\_increment**

The window size increment.

**Type**

int32\_t

**reserved**

Reserved bit. Currently this is always set to 0 and application should not expect something useful in here.

**Type**

uint8\_t

**hd**

Structure/Union member

**reserved**

Structure/Union member

**window\_size\_increment**

Structure/Union member

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### p

- `pynghttp2`, [5](#)
- `pynghttp2.http2`, [5](#)
- `pynghttp2.messages`, [7](#)
- `pynghttp2.nghttp2`, [9](#)
- `pynghttp2.sessions`, [5](#)
- `pynghttp2.streams`, [8](#)
- `pynghttp2.typedefs`, [14](#)





## A

ACK (*pynghttp2.typedefs.flag attribute*), 19  
age (*pynghttp2.typedefs.info attribute*), 25  
ALTSVC (*pynghttp2.typedefs.frame\_type attribute*), 22  
at\_eof() (*pynghttp2.streams.StreamReader method*), 8  
authority (*pynghttp2.messages.HTTP2Message property*), 7

## B

BAD\_CLIENT\_MAGIC (*pynghttp2.typedefs.error attribute*), 15  
BaseHTTP2 (*class in pynghttp2.sessions*), 5  
begin\_headers() (*pynghttp2.sessions.ClientProtocol method*), 6  
begin\_headers() (*pynghttp2.sessions.ServerProtocol method*), 7  
BUFFER\_ERROR (*pynghttp2.typedefs.error attribute*), 15

## C

CALLBACK\_FAILURE (*pynghttp2.typedefs.error attribute*), 15  
can\_write\_stream() (*pynghttp2.sessions.BaseHTTP2 method*), 5  
CANCEL (*pynghttp2.typedefs.error attribute*), 15  
CANCEL (*pynghttp2.typedefs.error\_code attribute*), 18  
cast\_py\_object() (*in module pynghttp2.nghttp2*), 14  
cat (*pynghttp2.typedefs.headers attribute*), 24  
CLIENT (*pynghttp2.nghttp2.session\_type attribute*), 14  
ClientProtocol (*class in pynghttp2.sessions*), 6  
ClientSession (*class in pynghttp2.sessions*), 7  
close() (*pynghttp2.sessions.ServerSession method*), 7  
CLOSED (*pynghttp2.typedefs.stream\_proto\_state attribute*), 31  
COMPRESSION\_ERROR (*pynghttp2.typedefs.error\_code attribute*), 18  
CONNECT\_ERROR (*pynghttp2.typedefs.error\_code attribute*), 18  
connection\_lost() (*pynghttp2.sessions.BaseHTTP2 method*), 5  
connection\_made() (*pynghttp2.sessions.BaseHTTP2 method*), 5  
consume() (*pynghttp2.nghttp2.Session method*), 11

consume\_connection() (*pynghttp2.nghttp2.Session method*), 11  
consume\_stream() (*pynghttp2.nghttp2.Session method*), 11  
content\_length (*pynghttp2.messages.HTTP2Message property*), 7  
content\_received() (*pynghttp2.sessions.ClientProtocol method*), 6  
content\_received() (*pynghttp2.sessions.ServerProtocol method*), 7  
content\_sent() (*pynghttp2.messages.HTTP2Message method*), 7  
content\_sent() (*pynghttp2.sessions.BaseHTTP2 method*), 5  
content\_type (*pynghttp2.messages.HTTP2Message property*), 8  
CONTINUATION (*pynghttp2.typedefs.frame\_type attribute*), 22

## D

data (*class in pynghttp2.typedefs*), 14  
data (*pynghttp2.typedefs.frame attribute*), 19, 20  
DATA (*pynghttp2.typedefs.frame\_type attribute*), 22  
DATA\_EXIST (*pynghttp2.typedefs.error attribute*), 16  
data\_flag (*class in pynghttp2.typedefs*), 14  
DATA\_MIN\_SIZE (*in module pynghttp2.streams*), 8  
data\_provider (*class in pynghttp2.typedefs*), 14  
data\_received() (*pynghttp2.sessions.BaseHTTP2 method*), 5  
data\_source (*class in pynghttp2.typedefs*), 15  
data\_source\_read\_callback (*in module pynghttp2.typedefs*), 15  
DEFERRED (*pynghttp2.typedefs.error attribute*), 16  
DEFERRED\_DATA\_EXIST (*pynghttp2.typedefs.error attribute*), 16  
Direction (*class in pynghttp2.messages*), 7  
drain() (*pynghttp2.sessions.BaseHTTP2 method*), 6  
drain() (*pynghttp2.streams.StreamReader method*), 8

## E

EMIT (*pynghttp2.typedefs.hd\_inflate\_flag* attribute), 24  
 empty() (*pynghttp2.streams.StreamReader* method), 8  
 ENABLE\_PUSH (*pynghttp2.typedefs.settings\_id* attribute), 31  
 END\_HEADERS (*pynghttp2.typedefs.flag* attribute), 19  
 END\_STREAM (*pynghttp2.typedefs.flag* attribute), 19  
 ENHANCE\_YOUR\_CALM (*pynghttp2.typedefs.error\_code* attribute), 18  
 EOF (*pynghttp2.typedefs.data\_flag* attribute), 14  
 EOF (*pynghttp2.typedefs.error* attribute), 16  
 error (*class in pynghttp2.typedefs*), 15  
 error\_code (*class in pynghttp2.typedefs*), 18  
 error\_code (*pynghttp2.typedefs.goaway* attribute), 23  
 error\_code (*pynghttp2.typedefs.rst\_stream* attribute), 29  
 establish\_session() (*pynghttp2.sessions.ClientProtocol* method), 6  
 establish\_session() (*pynghttp2.sessions.ServerProtocol* method), 7  
 exclusive (*pynghttp2.typedefs.priority\_spec* attribute), 28  
 ext (*pynghttp2.typedefs.frame* attribute), 20  
 extension (*class in pynghttp2.typedefs*), 18

## F

FATAL (*pynghttp2.typedefs.error* attribute), 16  
 fd (*pynghttp2.typedefs.data\_source* attribute), 15  
 feed\_data() (*pynghttp2.streams.StreamReader* method), 9  
 feed\_eof() (*pynghttp2.streams.StreamReader* method), 9  
 FINAL (*pynghttp2.typedefs.hd\_inflate\_flag* attribute), 24  
 flag (*class in pynghttp2.typedefs*), 19  
 flags (*pynghttp2.typedefs.frame\_hd* attribute), 21, 22  
 flags (*pynghttp2.typedefs.nv* attribute), 26  
 FLOODED (*pynghttp2.typedefs.error* attribute), 16  
 FLOW\_CONTROL (*pynghttp2.typedefs.error* attribute), 16  
 FLOW\_CONTROL\_ERROR (*pynghttp2.typedefs.error\_code* attribute), 18  
 flush() (*pynghttp2.sessions.BaseHTTP2* method), 6  
 frame (*class in pynghttp2.typedefs*), 19  
 frame\_hd (*class in pynghttp2.typedefs*), 21  
 FRAME\_SIZE\_ERROR (*pynghttp2.typedefs.error* attribute), 16  
 FRAME\_SIZE\_ERROR (*pynghttp2.typedefs.error\_code* attribute), 18  
 frame\_type (*class in pynghttp2.typedefs*), 22

## G

get() (*in module pynghttp2.http2*), 5

get() (*pynghttp2.sessions.ClientSession* method), 7  
 get\_local\_settings() (*pynghttp2.nghttp2.Session* method), 12  
 get\_local\_window\_size() (*pynghttp2.nghttp2.Session* method), 12  
 get\_remote\_settings() (*pynghttp2.nghttp2.Session* method), 12  
 get\_remote\_window\_size() (*pynghttp2.nghttp2.Session* method), 12  
 get\_stream\_local\_close() (*pynghttp2.nghttp2.Session* method), 12  
 get\_stream\_local\_window\_size() (*pynghttp2.nghttp2.Session* method), 12  
 get\_stream\_remote\_close() (*pynghttp2.nghttp2.Session* method), 12  
 get\_stream\_remote\_window\_size() (*pynghttp2.nghttp2.Session* method), 13  
 get\_stream\_user\_data() (*pynghttp2.nghttp2.Session* method), 13  
 goaway (*class in pynghttp2.typedefs*), 22  
 goaway (*pynghttp2.typedefs.frame* attribute), 20  
 GOAWAY (*pynghttp2.typedefs.frame\_type* attribute), 22  
 GOAWAY\_ALREADY\_SENT (*pynghttp2.typedefs.error* attribute), 16  
 goaway\_received() (*pynghttp2.sessions.BaseHTTP2* method), 6  
 goaway\_sent() (*pynghttp2.sessions.BaseHTTP2* method), 6

## H

HALF\_CLOSED\_LOCAL (*pynghttp2.typedefs.stream\_proto\_state* attribute), 31  
 HALF\_CLOSED\_REMOTE (*pynghttp2.typedefs.stream\_proto\_state* attribute), 31  
 hd (*pynghttp2.typedefs.extension* attribute), 18, 19  
 hd (*pynghttp2.typedefs.frame* attribute), 19, 21  
 hd (*pynghttp2.typedefs.goaway* attribute), 23  
 hd (*pynghttp2.typedefs.headers* attribute), 24  
 hd (*pynghttp2.typedefs.ping* attribute), 27  
 hd (*pynghttp2.typedefs.priority* attribute), 27  
 hd (*pynghttp2.typedefs.push\_promise* attribute), 28, 29  
 hd (*pynghttp2.typedefs.rst\_stream* attribute), 29  
 hd (*pynghttp2.typedefs.settings* attribute), 30  
 hd (*pynghttp2.typedefs.window\_update* attribute), 32  
 hd\_inflate\_flag (*class in pynghttp2.typedefs*), 23  
 HEADER\_COMP (*pynghttp2.typedefs.error* attribute), 16  
 HEADER\_TABLE\_SIZE (*pynghttp2.typedefs.settings\_id* attribute), 31  
 headers (*class in pynghttp2.typedefs*), 24  
 headers (*pynghttp2.typedefs.frame* attribute), 19, 21  
 HEADERS (*pynghttp2.typedefs.frame\_type* attribute), 22

HEADERS (pynghttp2.typedefs.headers\_category attribute), 25

headers\_category (class in pynghttp2.typedefs), 25

headers\_received() (pynghttp2.messages.HTTP2Message method), 8

headers\_received() (pynghttp2.sessions.ClientProtocol method), 6

headers\_received() (pynghttp2.sessions.ServerProtocol method), 7

headers\_sent() (pynghttp2.messages.HTTP2Message method), 8

headers\_sent() (pynghttp2.sessions.BaseHTTP2 method), 6

HTTP2Message (class in pynghttp2.messages), 7

HTTP\_1\_1\_REQUIRED (pynghttp2.typedefs.error\_code attribute), 18

HTTP\_HEADER (pynghttp2.typedefs.error attribute), 16

HTTP\_MESSAGING (pynghttp2.typedefs.error attribute), 16

I

IDLE (pynghttp2.typedefs.stream\_proto\_state attribute), 31

INADEQUATE\_SECURITY (pynghttp2.typedefs.error\_code attribute), 18

info (class in pynghttp2.typedefs), 25

INITIAL\_WINDOW\_SIZE (pynghttp2.typedefs.settings\_id attribute), 31

INSUFF\_BUFSIZE (pynghttp2.typedefs.error attribute), 16

INTERNAL (pynghttp2.typedefs.error attribute), 16

INTERNAL\_ERROR (pynghttp2.typedefs.error\_code attribute), 18

INVALID\_ARGUMENT (pynghttp2.typedefs.error attribute), 16

INVALID\_FRAME (pynghttp2.typedefs.error attribute), 16

INVALID\_HEADER\_BLOCK (pynghttp2.typedefs.error attribute), 16

INVALID\_STATE (pynghttp2.typedefs.error attribute), 17

INVALID\_STREAM\_ID (pynghttp2.typedefs.error attribute), 17

INVALID\_STREAM\_STATE (pynghttp2.typedefs.error attribute), 17

is\_deferred() (pynghttp2.streams.StreamReader method), 9

is\_eof() (pynghttp2.streams.StreamReader method), 9

is\_open() (pynghttp2.nghttp2.Session method), 13

iv (pynghttp2.typedefs.settings attribute), 30

## J

json() (pynghttp2.messages.HTTP2Message method), 8

## L

last\_stream\_id (pynghttp2.typedefs.goaway attribute), 23

length (pynghttp2.typedefs.frame\_hd attribute), 21, 22

## M

MAX\_CONCURRENT\_STREAMS (pynghttp2.typedefs.settings\_id attribute), 31

MAX\_FRAME\_SIZE (pynghttp2.typedefs.settings\_id attribute), 31

MAX\_HEADER\_LIST\_SIZE (pynghttp2.typedefs.settings\_id attribute), 31

mem\_recv() (pynghttp2.nghttp2.Session method), 13

mem\_send() (pynghttp2.nghttp2.Session method), 13

method (pynghttp2.messages.HTTP2Message property), 8

method (pynghttp2.messages.Request property), 8

module

pynghttp2, 5

pynghttp2.http2, 5

pynghttp2.messages, 7

pynghttp2.nghttp2, 9

pynghttp2.sessions, 5

pynghttp2.streams, 8

pynghttp2.typedefs, 14

## N

name (pynghttp2.typedefs.nv attribute), 26

namelen (pynghttp2.typedefs.nv attribute), 26

niv (pynghttp2.typedefs.settings attribute), 30

NO\_COPY (pynghttp2.typedefs.data\_flag attribute), 14

NO\_COPY\_NAME (pynghttp2.typedefs.nv\_flag attribute), 26

NO\_COPY\_VALUE (pynghttp2.typedefs.nv\_flag attribute), 26

NO\_END\_STREAM (pynghttp2.typedefs.data\_flag attribute), 14

NO\_ERROR (pynghttp2.typedefs.error\_code attribute), 18

NO\_INDEX (pynghttp2.typedefs.nv\_flag attribute), 26

NOMEM (pynghttp2.typedefs.error attribute), 17

NONE (pynghttp2.typedefs.data\_flag attribute), 14

NONE (pynghttp2.typedefs.flag attribute), 19

NONE (pynghttp2.typedefs.hd\_inflate\_flag attribute), 24

NONE (pynghttp2.typedefs.nv\_flag attribute), 26

nv (class in pynghttp2.typedefs), 25

nv\_flag (class in pynghttp2.typedefs), 26

nva (pynghttp2.typedefs.headers attribute), 24

nva (pynghttp2.typedefs.push\_promise attribute), 28, 29

nvlen (pynghttp2.typedefs.headers attribute), 24, 25

nvlen (pynghttp2.typedefs.push\_promise attribute), 28, 29

## O

on\_begin\_headers\_callback (in module pynghttp2.typedefs), 26

[on\\_data\\_chunk\\_recv\(\)](#) (*pynghttp2.sessions.BaseHTTP2 method*), 6  
[on\\_data\\_chunk\\_recv\\_callback](#) (*in module pynghttp2.typedefs*), 27  
[on\\_frame\\_recv\\_callback](#) (*in module pynghttp2.typedefs*), 27  
[on\\_frame\\_send\\_callback](#) (*in module pynghttp2.typedefs*), 27  
[on\\_header\(\)](#) (*pynghttp2.sessions.BaseHTTP2 method*), 6  
[on\\_header\\_callback](#) (*in module pynghttp2.typedefs*), 27  
[on\\_stream\\_close\\_callback](#) (*in module pynghttp2.typedefs*), 27  
[opaque\\_data](#) (*pynghttp2.typedefs.goaway attribute*), 23  
[opaque\\_data](#) (*pynghttp2.typedefs.ping attribute*), 27  
[opaque\\_data\\_len](#) (*pynghttp2.typedefs.goaway attribute*), 23  
[OPEN](#) (*pynghttp2.typedefs.stream\_proto\_state attribute*), 31  
[option\\_p](#) (*in module pynghttp2.typedefs*), 27  
[Options](#) (*class in pynghttp2.nghttp2*), 9

## P

[PADDED](#) (*pynghttp2.typedefs.flag attribute*), 19  
[padlen](#) (*pynghttp2.typedefs.data attribute*), 14  
[padlen](#) (*pynghttp2.typedefs.headers attribute*), 24, 25  
[padlen](#) (*pynghttp2.typedefs.push\_promise attribute*), 28, 29  
[path](#) (*pynghttp2.messages.HTTP2Message property*), 8  
[path](#) (*pynghttp2.messages.Request property*), 8  
[PAUSE](#) (*pynghttp2.typedefs.error attribute*), 17  
[pause\\_writing\(\)](#) (*pynghttp2.sessions.BaseHTTP2 method*), 6  
[payload](#) (*pynghttp2.typedefs.extension attribute*), 19  
[ping](#) (*class in pynghttp2.typedefs*), 27  
[ping](#) (*pynghttp2.typedefs.frame attribute*), 20, 21  
[PING](#) (*pynghttp2.typedefs.frame\_type attribute*), 22  
[post\(\)](#) (*in module pynghttp2.http2*), 5  
[post\(\)](#) (*pynghttp2.sessions.ClientSession method*), 7  
[pri\\_spec](#) (*pynghttp2.typedefs.headers attribute*), 24, 25  
[pri\\_spec](#) (*pynghttp2.typedefs.priority attribute*), 27, 28  
[priority](#) (*class in pynghttp2.typedefs*), 27  
[PRIORITY](#) (*pynghttp2.typedefs.flag attribute*), 19  
[priority](#) (*pynghttp2.typedefs.frame attribute*), 20, 21  
[PRIORITY](#) (*pynghttp2.typedefs.frame\_type attribute*), 22  
[priority\\_spec](#) (*class in pynghttp2.typedefs*), 28  
[promised\\_stream\\_id](#) (*pynghttp2.typedefs.push\_promise attribute*), 29  
[PROTO](#) (*pynghttp2.typedefs.error attribute*), 17  
[proto](#) (*pynghttp2.typedefs.info property*), 25  
[proto\\_str](#) (*pynghttp2.typedefs.info attribute*), 25

[PROTOCOL\\_ERROR](#) (*pynghttp2.typedefs.error\_code attribute*), 18  
[ptr](#) (*pynghttp2.typedefs.data\_source attribute*), 15  
[PUSH\\_DISABLED](#) (*pynghttp2.typedefs.error attribute*), 17  
[push\\_promise](#) (*class in pynghttp2.typedefs*), 28  
[push\\_promise](#) (*pynghttp2.typedefs.frame attribute*), 20, 21  
[PUSH\\_PROMISE](#) (*pynghttp2.typedefs.frame\_type attribute*), 22  
[PUSH\\_RESPONSE](#) (*pynghttp2.typedefs.headers\_category attribute*), 25  
[pynghttp2](#)  
     *module*, 5  
[pynghttp2.http2](#)  
     *module*, 5  
[pynghttp2.messages](#)  
     *module*, 7  
[pynghttp2.nghttp2](#)  
     *module*, 9  
[pynghttp2.sessions](#)  
     *module*, 5  
[pynghttp2.streams](#)  
     *module*, 8  
[pynghttp2.typedefs](#)  
     *module*, 14

## R

[read\(\)](#) (*pynghttp2.messages.HTTP2Message method*), 8  
[read\(\)](#) (*pynghttp2.streams.StreamReader method*), 9  
[read\\_callback](#) (*pynghttp2.typedefs.data\_provider attribute*), 15  
[read\\_nowait\(\)](#) (*pynghttp2.streams.StreamReader method*), 9  
[readany\(\)](#) (*pynghttp2.streams.StreamReader method*), 9  
[reading\\_deferred\(\)](#) (*pynghttp2.streams.StreamReader method*), 9  
[reading\\_resumed\(\)](#) (*pynghttp2.streams.StreamReader method*), 9  
[RECEIVING](#) (*pynghttp2.messages.Direction attribute*), 7  
[REFUSED\\_STREAM](#) (*pynghttp2.typedefs.error attribute*), 17  
[REFUSED\\_STREAM](#) (*pynghttp2.typedefs.error\_code attribute*), 18  
[Request](#) (*class in pynghttp2.messages*), 8  
[REQUEST](#) (*pynghttp2.typedefs.headers\_category attribute*), 25  
[request\(\)](#) (*in module pynghttp2.http2*), 5  
[request\(\)](#) (*pynghttp2.sessions.ClientSession method*), 7  
[request\\_allowed\(\)](#) (*pynghttp2.nghttp2.Session method*), 13  
[request\\_allowed\(\)](#) (*pynghttp2.sessions.ClientSession method*), 7  
[reserved](#) (*pynghttp2.typedefs.frame\_hd attribute*), 21, 22



- reserved (*pynghttp2.typedefs.goaway attribute*), 23
- reserved (*pynghttp2.typedefs.push\_promise attribute*), 29
- reserved (*pynghttp2.typedefs.window\_update attribute*), 32
- RESERVED\_LOCAL (*pynghttp2.typedefs.stream\_proto\_state attribute*), 31
- RESERVED\_REMOTE (*pynghttp2.typedefs.stream\_proto\_state attribute*), 31
- Response (*class in pynghttp2.messages*), 8
- RESPONSE (*pynghttp2.typedefs.headers\_category attribute*), 25
- response() (*pynghttp2.messages.Request method*), 8
- resume\_data() (*pynghttp2.nghttp2.Session method*), 13
- resume\_writing() (*pynghttp2.sessions.BaseHTTP2 method*), 6
- rst\_stream (*class in pynghttp2.typedefs*), 29
- rst\_stream (*pynghttp2.typedefs.frame attribute*), 20, 21
- RST\_STREAM (*pynghttp2.typedefs.frame\_type attribute*), 22
- ## S
- scheme (*pynghttp2.messages.HTTP2Message property*), 8
- send() (*pynghttp2.nghttp2.Session method*), 13
- send\_callback (*in module pynghttp2.typedefs*), 29
- send\_data\_callback (*in module pynghttp2.typedefs*), 30
- SENDING (*pynghttp2.messages.Direction attribute*), 7
- SERVER (*pynghttp2.nghttp2.session\_type attribute*), 14
- ServerProtocol (*class in pynghttp2.sessions*), 7
- ServerSession (*class in pynghttp2.sessions*), 7
- Session (*class in pynghttp2.nghttp2*), 11
- session\_callbacks\_p (*in module pynghttp2.typedefs*), 30
- SESSION\_CLOSING (*pynghttp2.typedefs.error attribute*), 17
- session\_get\_stream\_user\_data() (*in module pynghttp2.nghttp2*), 14
- session\_p (*in module pynghttp2.typedefs*), 30
- session\_set\_stream\_user\_data() (*in module pynghttp2.nghttp2*), 14
- session\_type (*class in pynghttp2.nghttp2*), 14
- set\_builtin\_recv\_extension\_type() (*pynghttp2.nghttp2.Options method*), 9
- set\_exception() (*pynghttp2.messages.HTTP2Message method*), 8
- set\_exception() (*pynghttp2.streams.StreamReader method*), 9
- set\_max\_deflate\_dynamic\_table\_size() (*pynghttp2.nghttp2.Options method*), 9
- set\_max\_reserved\_remote\_streams() (*pynghttp2.nghttp2.Options method*), 10
- set\_max\_send\_header\_block\_length() (*pynghttp2.nghttp2.Options method*), 10
- set\_no\_auto\_ping\_ack() (*pynghttp2.nghttp2.Options method*), 10
- set\_no\_auto\_window\_update() (*pynghttp2.nghttp2.Options method*), 10
- set\_no\_closed\_streams() (*pynghttp2.nghttp2.Options method*), 10
- set\_no\_http\_messaging() (*pynghttp2.nghttp2.Options method*), 10
- set\_no\_recv\_client\_magic() (*pynghttp2.nghttp2.Options method*), 10
- set\_peer\_max\_concurrent\_streams() (*pynghttp2.nghttp2.Options method*), 11
- set\_protocol() (*pynghttp2.streams.StreamReader method*), 9
- set\_stream\_user\_data() (*pynghttp2.nghttp2.Session method*), 13
- set\_user\_recv\_extension\_type() (*pynghttp2.nghttp2.Options method*), 11
- settings (*class in pynghttp2.typedefs*), 30
- settings (*pynghttp2.typedefs.frame attribute*), 20, 21
- SETTINGS (*pynghttp2.typedefs.frame\_type attribute*), 22
- settings\_entry (*class in pynghttp2.typedefs*), 30
- SETTINGS\_EXPECTED (*pynghttp2.typedefs.error attribute*), 17
- settings\_id (*class in pynghttp2.typedefs*), 31
- settings\_id (*pynghttp2.typedefs.settings\_entry attribute*), 30
- SETTINGS\_TIMEOUT (*pynghttp2.typedefs.error\_code attribute*), 18
- settings\_updated() (*pynghttp2.sessions.BaseHTTP2 method*), 6
- source (*pynghttp2.typedefs.data\_provider attribute*), 14, 15
- start() (*pynghttp2.sessions.ClientSession method*), 7
- start() (*pynghttp2.sessions.ServerSession method*), 7
- START\_STREAM\_NOT\_ALLOWED (*pynghttp2.typedefs.error attribute*), 17
- status (*pynghttp2.messages.Response property*), 8
- STREAM\_CLOSED (*pynghttp2.typedefs.error attribute*), 17
- STREAM\_CLOSED (*pynghttp2.typedefs.error\_code attribute*), 18
- stream\_closed() (*pynghttp2.messages.HTTP2Message method*), 8
- stream\_closed() (*pynghttp2.sessions.BaseHTTP2 method*), 6
- stream\_closed() (*pynghttp2.sessions.ClientProtocol method*), 6
- STREAM\_CLOSING (*pynghttp2.typedefs.error attribute*), 17

`stream_exists()` (*pynghttp2.nghttp2.Session method*), 13  
`stream_id` (*pynghttp2.messages.HTTP2Message property*), 8  
`stream_id` (*pynghttp2.typedefs.frame\_hd attribute*), 21, 22  
`stream_id` (*pynghttp2.typedefs.priority\_spec attribute*), 28  
`STREAM_ID_NOT_AVAILABLE` (*pynghttp2.typedefs.error attribute*), 17  
`stream_p` (*in module pynghttp2.typedefs*), 31  
`stream_proto_state` (*class in pynghttp2.typedefs*), 31  
`STREAM_SHUT_WR` (*pynghttp2.typedefs.error attribute*), 17  
`StreamClosedError`, 8  
`StreamReader` (*class in pynghttp2.streams*), 8  
`submit_data()` (*pynghttp2.nghttp2.Session method*), 13  
`submit_headers()` (*pynghttp2.nghttp2.Session method*), 13  
`submit_request()` (*pynghttp2.nghttp2.Session method*), 13  
`submit_request()` (*pynghttp2.sessions.ClientProtocol method*), 7  
`submit_response()` (*pynghttp2.nghttp2.Session method*), 13  
`submit_response()` (*pynghttp2.sessions.BaseHTTP2 method*), 6  
`submit_settings()` (*pynghttp2.nghttp2.Session method*), 13

## T

`TEMPORAL_CALLBACK_FAILURE` (*pynghttp2.typedefs.error attribute*), 17  
`terminate()` (*pynghttp2.nghttp2.Session method*), 13  
`terminate()` (*pynghttp2.sessions.BaseHTTP2 method*), 6  
`terminate()` (*pynghttp2.sessions.ClientSession method*), 7  
`text()` (*pynghttp2.messages.HTTP2Message method*), 8  
`TOO_MANY_INFLIGHT_SETTINGS` (*pynghttp2.typedefs.error attribute*), 17  
`type` (*pynghttp2.typedefs.frame\_hd attribute*), 21, 22

## U

`UNSUPPORTED_VERSION` (*pynghttp2.typedefs.error attribute*), 17

## V

`value` (*pynghttp2.typedefs.nv attribute*), 26  
`value` (*pynghttp2.typedefs.settings\_entry attribute*), 30, 31  
`valuelen` (*pynghttp2.typedefs.nv attribute*), 26  
`version` (*pynghttp2.typedefs.info property*), 25  
`version()` (*in module pynghttp2.nghttp2*), 14

`version_info` (*pynghttp2.typedefs.info property*), 25  
`version_num` (*pynghttp2.typedefs.info attribute*), 25  
`version_str` (*pynghttp2.typedefs.info attribute*), 25

## W

`wait_closed()` (*pynghttp2.sessions.ServerSession method*), 7  
`wait_eof()` (*pynghttp2.streams.StreamReader method*), 9  
`wait_for_window_update()` (*pynghttp2.sessions.BaseHTTP2 method*), 6  
`want_read()` (*pynghttp2.nghttp2.Session method*), 13  
`want_write()` (*pynghttp2.nghttp2.Session method*), 13  
`weight` (*pynghttp2.typedefs.priority\_spec attribute*), 28  
`window_size_increment` (*pynghttp2.typedefs.window\_update attribute*), 32  
`window_update` (*class in pynghttp2.typedefs*), 31  
`window_update` (*pynghttp2.typedefs.frame attribute*), 20, 21  
`WINDOW_UPDATE` (*pynghttp2.typedefs.frame\_type attribute*), 22  
`window_update_received()` (*pynghttp2.sessions.BaseHTTP2 method*), 6  
`WOULDBLOCK` (*pynghttp2.typedefs.error attribute*), 18