pynghttp2 Documentation

Release 0.1.0

Lucas Kahlert

Jun 01, 2023

CONTENTS

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.

CHAPTER

ONE

QUICKSTART

CHAPTER

TWO

API

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_recv_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_recv_client_magic=None*, *peer_max_concurrent_streams=None*, *user_recv_extension_type=None*)

Bases: object

set_builtin_recv_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_recv_extension_type() and nghttp2_option_set_user_recv_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_recv(*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_recv_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.**info**

Bases: 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.**nv**

Bases: 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_recv_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_recv_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_recv_callback**
alias of CFunctionType

pynghttp2.typedefs.**on_frame_recv_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

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W