

NSB

0.1.0

Generated by Doxygen 1.14.0

| | |
|---|----------|
| 1 Namespace Index | 1 |
| 1.1 Namespace List | 1 |
| 2 Hierarchical Index | 1 |
| 2.1 Class Hierarchy | 1 |
| 3 Class Index | 1 |
| 3.1 Class List | 1 |
| 4 File Index | 2 |
| 4.1 File List | 2 |
| 5 Namespace Documentation | 2 |
| 5.1 nsb_client Namespace Reference | 2 |
| 5.1.1 Function Documentation | 3 |
| 5.1.2 Variable Documentation | 3 |
| 6 Class Documentation | 4 |
| 6.1 nsb_client.CommunicationInterface Class Reference | 4 |
| 6.1.1 Detailed Description | 4 |
| 6.2 MessageEntry Struct Reference | 4 |
| 6.2.1 Detailed Description | 5 |
| 6.2.2 Constructor & Destructor Documentation | 5 |
| 6.2.3 Member Data Documentation | 5 |
| 6.3 nsb_client.NSAppClient Class Reference | 6 |
| 6.3.1 Detailed Description | 7 |
| 6.3.2 Constructor & Destructor Documentation | 7 |
| 6.3.3 Member Function Documentation | 7 |
| 6.3.4 Member Data Documentation | 8 |
| 6.4 nsb_client.NSBClient Class Reference | 8 |
| 6.4.1 Detailed Description | 9 |
| 6.4.2 Constructor & Destructor Documentation | 9 |
| 6.4.3 Member Function Documentation | 9 |
| 6.4.4 Member Data Documentation | 10 |
| 6.5 NSBDaemon Class Reference | 10 |
| 6.5.1 Constructor & Destructor Documentation | 11 |
| 6.5.2 Member Function Documentation | 12 |
| 6.5.3 Member Data Documentation | 16 |
| 6.6 nsb_client.NSBSimClient Class Reference | 17 |
| 6.6.1 Detailed Description | 18 |
| 6.6.2 Constructor & Destructor Documentation | 18 |
| 6.6.3 Member Function Documentation | 19 |
| 6.6.4 Member Data Documentation | 20 |
| 6.7 nsb_client.SocketInterface Class Reference | 20 |

| | |
|--|-----------|
| 6.7.1 Detailed Description | 21 |
| 6.7.2 Constructor & Destructor Documentation | 21 |
| 6.7.3 Member Function Documentation | 21 |
| 6.7.4 Member Data Documentation | 23 |
| 7 File Documentation | 23 |
| 7.1 nsb_client.py File Reference | 23 |
| 7.2 nsb_daemon.cc File Reference | 24 |
| 7.2.1 Function Documentation | 24 |
| 7.3 nsb_daemon.h File Reference | 25 |
| 7.3.1 Variable Documentation | 25 |
| 7.4 nsb_daemon.h | 26 |
| Index | 27 |

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

| | |
|----------------------------|---|
| nsb_client | 2 |
|----------------------------|---|

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

| | |
|--|-----------|
| nsb_client.CommunicationInterface | 4 |
| nsb_client.SocketInterface | 20 |
| MessageEntry | 4 |
| nsb_client.NSBClient | 8 |
| nsb_client.NSBAppClient | 6 |
| nsb_client.NSBSimClient | 17 |
| NSBDaemon | 10 |

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

| | |
|---|----|
| nsb_client.CommunicationInterface | |
| Base class for communication interfaces | 4 |
| MessageEntry | |
| Message storage struct | 4 |
| nsb_client.NSBAppClient | |
| NSB Application Client interface | 6 |
| nsb_client.NSBClient | |
| NSB client base class | 8 |
| NSBDaemon | 10 |
| nsb_client.NSBSimClient | |
| NSB Simulator Client interface | 17 |
| nsb_client.SocketInterface | |
| Socket interface for client-server communication | 20 |

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

| | |
|-------------------------------|----|
| nsb_client.py | 23 |
| nsb_daemon.cc | 24 |
| nsb_daemon.h | 25 |

5 Namespace Documentation

5.1 nsb_client Namespace Reference

Classes

- class [CommunicationInterface](#)
Base class for communication interfaces.
- class [NSBAppClient](#)
NSB Application Client interface.
- class [NSBClient](#)
NSB client base class.
- class [NSBSimClient](#)
NSB Simulator Client interface.
- class [SocketInterface](#)
Socket interface for client-server communication.

Functions

- [test_ping](#) ()
- [test_lifecycle](#) ()

Variables

- int [SERVER_CONNECTION_TIMEOUT](#) = 10
- int [DAEMON_RESPONSE_TIMEOUT](#) = 600
- int [RECEIVE_BUFFER_SIZE](#) = 4096
Buffer size when receiving data.
- int [SEND_BUFFER_SIZE](#) = 4096
Buffer size when sending data.

5.1.1 Function Documentation

test_lifecycle()

```
nsb_client.test_lifecycle ()
```

test_ping()

```
nsb_client.test_ping ()
```

5.1.2 Variable Documentation

DAEMON_RESPONSE_TIMEOUT

```
int nsb_client.DAEMON_RESPONSE_TIMEOUT = 600
```

Maximum time a client will wait to get a response from the daemon.

RECEIVE_BUFFER_SIZE

```
int nsb_client.RECEIVE_BUFFER_SIZE = 4096
```

Buffer size when receiving data.

SEND_BUFFER_SIZE

```
int nsb_client.SEND_BUFFER_SIZE = 4096
```

Buffer size when sending data.

SERVER_CONNECTION_TIMEOUT

```
int nsb_client.SERVER_CONNECTION_TIMEOUT = 10
```

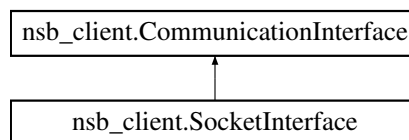
Maximum time a client will wait to connect to the daemon.

6 Class Documentation

6.1 nsb_client.CommunicationInterface Class Reference

Base class for communication interfaces.

Inheritance diagram for nsb_client.CommunicationInterface:



6.1.1 Detailed Description

Base class for communication interfaces.

As NSB is expected to support different communication paradigms and protocols – including sockets, RabbitMQ, and Websockets – we plan to provide various different communication interfaces with the same basic functions. The [SocketInterface](#) class should be used as an example to develop other interfaces.

The documentation for this class was generated from the following file:

- [nsb_client.py](#)

6.2 MessageEntry Struct Reference

Message storage struct.

```
#include <nsb_daemon.h>
```

Public Member Functions

- [MessageEntry](#) ()
Blank constructor.
- [MessageEntry](#) (std::string src, std::string dest, std::string data)
Populated constructor.

Public Attributes

- `std::string` [source](#)
The source identifier.
- `std::string` [destination](#)
The destination identifier.
- `std::string` [payload](#)
The payload as a bytestring, but in `const char` form.*

6.2.1 Detailed Description

Message storage struct.

This struct contains source and destination information and the payload and is intended to be used to store messages in the daemon's transmission and reception buffers.

6.2.2 Constructor & Destructor Documentation

MessageEntry() [1/2]

```
MessageEntry::MessageEntry () [inline]
```

Blank constructor.

MessageEntry() [2/2]

```
MessageEntry::MessageEntry (  
    std::string src,  
    std::string dest,  
    std::string data) [inline]
```

Populated constructor.

6.2.3 Member Data Documentation

destination

```
std::string MessageEntry::destination
```

The destination identifier.

payload

```
std::string MessageEntry::payload
```

The payload as a bytestring, but in `const char*` form.

source

```
std::string MessageEntry::source
```

The source identifier.

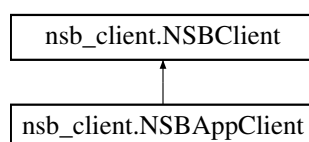
The documentation for this struct was generated from the following file:

- [nsb_daemon.h](#)

6.3 nsb_client.NSBAppClient Class Reference

NSB Application Client interface.

Inheritance diagram for nsb_client.NSBAppClient:

**Public Member Functions**

- [__init__](#) (self, str identifier, str server_address, int server_port)
Constructs the NSB Application Client interface.
- [send](#) (self, str dest_id, bytes payload)
Sends a payload to the specified destination via NSB.
- [receive](#) (self, str|None dest_id=None)
Receives a payload that has been addressed to the client via NSB.

Public Member Functions inherited from [nsb_client.NSBClient](#)

- [__init__](#) (self, str server_address, int server_port)
Base constructor for NSB Clients.
- [ping](#) (self, int timeout=DAEMON_RESPONSE_TIMEOUT)
Pings the server.
- [exit](#) (self)
Instructs server and self to exit and shutdown.

Public Attributes

- [logger](#) = logging.getLogger(f'{self._id} (app)')

Public Attributes inherited from [nsb_client.NSBClient](#)

- [comms](#) = [SocketInterface](#)(server_address, server_port)

Protected Attributes

- `_id` = identifier

6.3.1 Detailed Description

NSB Application Client interface.

This client provides the high-level NSB interface to send and receive messages via NSB by communicating to the daemon.

6.3.2 Constructor & Destructor Documentation

`__init__()`

```
nsb_client.NSBAppClient.__init__ (
    self,
    str identifier,
    str server_address,
    int server_port)
```

Constructs the NSB Application Client interface.

This method uses the base [NSBClient](#)'s constructor, which initializes a network interface to connect and communicate with the NSB daemon. It also an identifier that should correspond to the identifier used in the NSB system.

Parameters

| | |
|-----------------------|---|
| <i>identifier</i> | The identifier for this NSB application client, which should correspond to the identifier in NSB and simulator. |
| <i>server_address</i> | The address of the NSB daemon. |
| <i>server_port</i> | The port of the NSB daemon. |

6.3.3 Member Function Documentation

`receive()`

```
nsb_client.NSBAppClient.receive (
    self,
    str|None dest_id = None)
```

Receives a payload that has been addressed to the client via NSB.

If the destination is specified, it will receive a payload for that destination. This method creates an NSB RECEIVE message with the appropriate information and payload and sends it to the daemon. It will then get a response that either contains a MESSAGE code and carries the retrieved payload or contains a NO_MESSAGE code. If a message is found, the entire NSB message is returned to provide access to the metadata.

Parameters

| | |
|----------------|---|
| <i>dest_id</i> | The identifier of the destination NSB client. The default None value will automatically assume the destination is self. |
|----------------|---|

Returns

nsb_pb2.nsbm|None The NSB message containing the received payload and metadata if a message is found, otherwise None.

send()

```
nsb_client.NSBAppClient.send (
    self,
    str dest_id,
    bytes payload)
```

Sends a payload to the specified destination via NSB.

This method creates an NSB SEND message with the appropriate information and payload and sends it to the daemon. It does not expect a response from the daemon.

Parameters

| | |
|----------------|---|
| <i>dest_id</i> | The identifier of the destination NSB client. |
| <i>payload</i> | The payload to send to the destination. |

6.3.4 Member Data Documentation**_id**

```
nsb_client.NSBAppClient._id = identifier [protected]
```

logger

```
nsb_client.NSBAppClient.logger = logging.getLogger(f"{self._id} (app)")
```

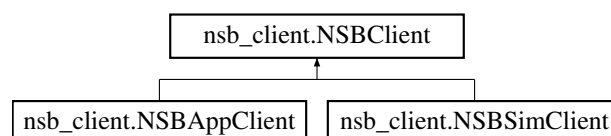
The documentation for this class was generated from the following file:

- [nsb_client.py](#)

6.4 nsb_client.NSBClient Class Reference

NSB client base class.

Inheritance diagram for nsb_client.NSBClient:



Public Member Functions

- `__init__` (self, str server_address, int server_port)
Base constructor for NSB Clients.
- `ping` (self, int timeout=DAEMON_RESPONSE_TIMEOUT)
Pings the server.
- `exit` (self)
Instructs server and self to exit and shutdown.

Public Attributes

- `comms` = [SocketInterface](#)(server_address, server_port)

6.4.1 Detailed Description

NSB client base class.

This class serves as the base for the implemented clients (AppClient and SimClient) will be built on. It provides basic methods and shared operation methods.

6.4.2 Constructor & Destructor Documentation

`__init__()`

```
nsb_client.NSBClient.__init__ (
    self,
    str server_address,
    int server_port)
```

Base constructor for NSB Clients.

Sets the communications module to the desired network interface (currently [SocketInterface](#)) to connect to the daemon server.

Parameters

| | |
|-----------------------------|--------------------------------|
| <code>server_address</code> | The address of the NSB daemon. |
| <code>server_port</code> | The port of the NSB daemon. |

See also

[SocketInterface](#)

6.4.3 Member Function Documentation

`exit()`

```
nsb_client.NSBClient.exit (
    self)
```

Instructs server and self to exit and shutdown.

This method sends an EXIT message to the server before deleting itself.

ping()

```
nsb_client.NSBClient.ping (
    self,
    int timeout = DAEMON_RESPONSE_TIMEOUT)
```

Pings the server.

This method sends a PING message to the server and awaits a response. It returns whether or not the server is reachable.

Parameters

| | |
|----------------|---|
| <i>timeout</i> | Maximum time to wait for a response from the server. Default is set to DAEMON_RESPONSE_TIMEOUT. |
|----------------|---|

Returns

bool True if the server is reachable and responds correctly, False otherwise.

6.4.4 Member Data Documentation

comms

```
nsb_client.NSBClient.comms = SocketInterface(server_address, server_port)
```

The documentation for this class was generated from the following file:

- [nsb_client.py](#)

6.5 NSBDaemon Class Reference

```
#include <nsb_daemon.h>
```

Public Member Functions

- [NSBDaemon](#) (int s_port)
Construct a new [NSBDaemon::NSBDaemon](#) object.
- [~NSBDaemon](#) ()
Destroy the [NSBDaemon::NSBDaemon](#) object.
- void [start](#) ()
Start the NSB Daemon.
- void [stop](#) ()
Stops the NSB Daemon.
- bool [is_running](#) () const
Checks if the server is running.

Private Member Functions

- void [start_server](#) (int port)
Start the socket-connected server within the NSB Daemon.
- void [handle_message](#) (int fd, std::vector< char > message)
A multiplexer to parse messages and redirect them to handlers.
- void [handle_ping](#) (nsb::nsbm *incoming_msg, nsb::nsbm *outgoing_msg, bool *response_required)
Handles PING messages.
- void [handle_send](#) (nsb::nsbm *incoming_msg, nsb::nsbm *outgoing_msg, bool *response_required)
Handles SEND messages from the NSB Application Client.
- void [handle_fetch](#) (nsb::nsbm *incoming_msg, nsb::nsbm *outgoing_msg, bool *response_required)
Handles FETCH messages from the NSB Simulator Client.
- void [handle_post](#) (nsb::nsbm *incoming_msg, nsb::nsbm *outgoing_msg, bool *response_required)
Handles POST messages from the NSB Simulator Client.
- void [handle_receive](#) (nsb::nsbm *incoming_msg, nsb::nsbm *outgoing_msg, bool *response_required)
Handles RECEIVE messages from the NSB Application Client.

Private Attributes

- std::atomic< bool > [running](#)
A flag set to indicate daemon server status.
- int [server_port](#)
The server port accessible to client connections.
- std::list< [MessageEntry](#) > [tx_buffer](#)
Transmission buffer to store sent payloads waiting to be fetched.
- std::list< [MessageEntry](#) > [rx_buffer](#)
Reception buffer to store posted payloads waiting to be received.

6.5.1 Constructor & Destructor Documentation

NSBDaemon()

```
NSBDaemon::NSBDaemon (
    int s_port)
```

Construct a new [NSBDaemon::NSBDaemon](#) object.

This method initializes attributes and verifies the Protobuf version.

Parameters

| | |
|------------------------|--|
| s_port | The port that NSB clients will connect to. |
|------------------------|--|

~NSBDaemon()

```
NSBDaemon::~NSBDaemon ()
```

Destroy the [NSBDaemon::NSBDaemon](#) object.

This method will check to see if the server is still running and stop it if necessary. It will then shut down the Protobuf library.

6.5.2 Member Function Documentation

handle_fetch()

```
void NSBDaemon::handle_fetch (
    nsb::nsbm * incoming_msg,
    nsb::nsbm * outgoing_msg,
    bool * response_required) [private]
```

Handles FETCH messages from the NSB Simulator Client.

This method first creates a blank [MessageEntry](#). If a source has been specified, it will search the transmission buffer for a message with that source, either setting the blank [MessageEntry](#) to the found entry if the query was resolved or leaving it blank if not found. If a source has not been specified, the top [MessageEntry](#) of the buffer will be popped off and used; otherwise, if the buffer is empty, the [MessageEntry](#) will be left blank.

If a message was found, a NSB FETCH message indicating MESSAGE will be sent with the metadata and payload. Otherwise, a NSB FETCH message indicating NO_MESSAGE will be sent back to the client.

Parameters

| | |
|--------------------------|---|
| <i>incoming_msg</i> | The incoming message that is being handled. |
| <i>outgoing_msg</i> | A template message that can be used if a response is required. |
| <i>response_required</i> | Whether or not a response is required and the outgoing message will be sent back to the client. |

See also

[MessageEntry](#)

handle_message()

```
void NSBDaemon::handle_message (
    int fd,
    std::vector< char > message) [private]
```

A multiplexer to parse messages and redirect them to handlers.

This method is invoked by the server (in the [start_server\(\)](#) method) to handle an incoming message. It parses the message using Protobuf, and then redirects the incoming message and a template outgoing message (in case a response is necessary) to one of the operation-specific handlers.

If the operation is not understood, the server will respond with a negative PING message.

Parameters

| | |
|----------------|---|
| <i>fd</i> | The file descriptor of the client connection. |
| <i>message</i> | The incoming message to parse and handle. |

See also

[start_server\(\)](#)
[handle_ping\(\)](#)
[handle_send\(\)](#)
[handle_fetch\(\)](#)
[handle_post\(\)](#)
[handle_receive\(\)](#)

handle_ping()

```
void NSBDaemon::handle_ping (
    nsb::nsbm * incoming_msg,
    nsb::nsbm * outgoing_msg,
    bool * response_required) [private]
```

Handles PING messages.

Since the PING has been received, it can be assumed to be successful. As such this method populates the outgoing message as an NSB PING message indicating success.

Parameters

| | |
|--------------------------|---|
| <i>incoming_msg</i> | The incoming message that is being handled. |
| <i>outgoing_msg</i> | A template message that can be used if a response is required. |
| <i>response_required</i> | Whether or not a response is required and the outgoing message will be sent back to the client. |

handle_post()

```
void NSBDaemon::handle_post (
    nsb::nsbm * incoming_msg,
    nsb::nsbm * outgoing_msg,
    bool * response_required) [private]
```

Handles POST messages from the NSB Simulator Client.

This method handles POST messages by parsing the incoming message and storing the source, destination, and payload as a [MessageEntry](#). The new [MessageEntry](#) will be pushed back in the reception buffer where it will be ready to be received by the NSB Application Client.

Parameters

| | |
|--------------------------|---|
| <i>incoming_msg</i> | The incoming message that is being handled. |
| <i>outgoing_msg</i> | A template message that can be used if a response is required. |
| <i>response_required</i> | Whether or not a response is required and the outgoing message will be sent back to the client. |

See also

[MessageEntry](#)

[handle_receive\(\)](#)

handle_receive()

```
void NSBDaemon::handle_receive (
    nsb::nsbm * incoming_msg,
    nsb::nsbm * outgoing_msg,
    bool * response_required) [private]
```

Handles RECEIVE messages from the NSB Application Client.

This method first creates a blank [MessageEntry](#). If a destination has been specified, it will search the reception buffer for a message with that destination, either setting the blank [MessageEntry](#) to the found entry if the query was resolved or leaving it blank if not found. If a destination has not been specified, the top [MessageEntry](#) of the buffer will be popped off and used; otherwise, if the buffer is empty, the [MessageEntry](#) will be left blank.

If a message was found, a NSB RECEIVE message indicating MESSAGE will be sent with the metadata and payload. Otherwise, a NSB RECEIVE message indicating NO_MESSAGE will be sent back to the client.

Parameters

| | |
|--------------------------|---|
| <i>incoming_msg</i> | The incoming message that is being handled. |
| <i>outgoing_msg</i> | A template message that can be used if a response is required. |
| <i>response_required</i> | Whether or not a response is required and the outgoing message will be sent back to the client. |

See also

[MessageEntry](#)**handle_send()**

```
void NSBDaemon::handle_send (
    nsb::nsbm * incoming_msg,
    nsb::nsbm * outgoing_msg,
    bool * response_required) [private]
```

Handles SEND messages from the NSB Application Client.

This method handles SEND messages by parsing the incoming message and storing the source, destination, and payload as a [MessageEntry](#). The new [MessageEntry](#) will be pushed back in the transmission buffer where it will be ready to be fetched by the NSB Simulator Client.

Parameters

| | |
|--------------------------|---|
| <i>incoming_msg</i> | The incoming message that is being handled. |
| <i>outgoing_msg</i> | A template message that can be used if a response is required. |
| <i>response_required</i> | Whether or not a response is required and the outgoing message will be sent back to the client. |

See also

[MessageEntry](#)[handle_fetch\(\)](#)**is_running()**

```
bool NSBDaemon::is_running () const
```

Checks if the server is running.

Returns

true if the server is running, false otherwise.

false if the server is not running.

start()

```
void NSBDaemon::start ()
```

Start the NSB Daemon.

This method will launch the server at the server port using the `start_server` method.

See also

[start_server\(int port\)](#)

start_server()

```
void NSBDaemon::start_server (
    int port) [private]
```

Start the socket-connected server within the NSB Daemon.

This is the main servicing method that runs for the lifetime of the NSB Daemon. It opens a multiple connection-enabled server and maintains persistent connections as communication channels with each NSB client that connects to it. New connections are managed through an updating vector of file descriptors where each represents a different connection. When messages come in from existing connections, they will be passed onto the `handle_message` method.

This method is invoked by the [start\(\)](#) method.

Parameters

| | |
|-------------|--|
| <i>port</i> | The port that will be accessible for clients to connect. |
|-------------|--|

See also

[start\(\)](#)

[handle_message\(\)](#)

stop()

```
void NSBDaemon::stop ()
```

Stops the NSB Daemon.

Checks if server is running and stops it, resulting in the daemon shutting down.

6.5.3 Member Data Documentation**running**

```
std::atomic<bool> NSBDaemon::running [private]
```

A flag set to indicate daemon server status.

rx_buffer

```
std::list<MessageEntry> NSBDaemon::rx_buffer [private]
```

Reception buffer to store posted payloads waiting to be received.

See also

[MessageEntry](#)
[handle_post\(\)](#)
[handle_receive\(\)](#)

server_port

```
int NSBDaemon::server_port [private]
```

The server port accessible to client connections.

tx_buffer

```
std::list<MessageEntry> NSBDaemon::tx_buffer [private]
```

Transmission buffer to store sent payloads waiting to be fetched.

See also

[MessageEntry](#)
[handle_send\(\)](#)
[handle_fetch\(\)](#)

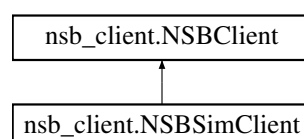
The documentation for this class was generated from the following files:

- [nsb_daemon.h](#)
- [nsb_daemon.cc](#)

6.6 nsb_client.NSBSimClient Class Reference

NSB Simulator Client interface.

Inheritance diagram for nsb_client.NSBSimClient:



Public Member Functions

- `__init__` (self, str server_address, int server_port)
Constructs the NSB Simulator Client interface.
- `fetch` (self, str|None src_id=None)
Fetches a payload that needs to be sent over the simulated network.
- `post` (self, str src_id, str dest_id, bytes payload, bool success=True)
Posts a payload to the specified destination via NSB.

Public Member Functions inherited from `nsb_client.NSBClient`

- `__init__` (self, str server_address, int server_port)
Base constructor for NSB Clients.
- `ping` (self, int timeout=DAEMON_RESPONSE_TIMEOUT)
Pings the server.
- `exit` (self)
Instructs server and self to exit and shutdown.

Public Attributes

- `logger` = logging.getLogger("(SimClient)")

Public Attributes inherited from `nsb_client.NSBClient`

- `comms` = `SocketInterface`(server_address, server_port)

6.6.1 Detailed Description

NSB Simulator Client interface.

This client provides the high-level NSB interface to fetch and post messages via NSB by communicating to the daemon.

6.6.2 Constructor & Destructor Documentation

`__init__()`

```
nsb_client.NSBClient.__init__ (
    self,
    str server_address,
    int server_port)
```

Constructs the NSB Simulator Client interface.

This method uses the base `NSBClient`'s constructor, which initializes a network interface to connect and communicate with the NSB daemon.

Parameters

| | |
|-----------------------|--------------------------------|
| <i>server_address</i> | The address of the NSB daemon. |
| <i>server_port</i> | The port of the NSB daemon. |

6.6.3 Member Function Documentation

fetch()

```
nsb_client.NSBSimClient.fetch (
    self,
    str|None src_id = None)
```

Fetches a payload that needs to be sent over the simulated network.

If the source is specified, it will try and fetch a payload for that source. This method creates an NSB FETCH message with the appropriate information and payload and sends it to the daemon. It will then get a response that either contains a MESSAGE code and carries the fetched payload or contains a NO_MESSAGE code. If a message is found, the entire NSB message is returned to provide access to the metadata.

Parameters

| | |
|---------------|--|
| <i>src_id</i> | The identifier of the target source. The default None value will result in fetching the most recent message, regardless of source. |
|---------------|--|

Returns

nsb_pb2.nsbm|None The NSB message containing the fetched payload and metadata if a message is found, otherwise None.

post()

```
nsb_client.NSBSimClient.post (
    self,
    str src_id,
    str dest_id,
    bytes payload,
    bool success = True)
```

Posts a payload to the specified destination via NSB.

This is intended to be used when a payload is finished being processed (either successfully delivered or dropped) and the simulator client needs to hand it off back to NSB. This method creates an NSB SEND message with the appropriate information and payload and sends it to the daemon.

Parameters

| | |
|----------------|---|
| <i>src_id</i> | The identifier of the source NSB client. |
| <i>dest_id</i> | The identifier of the destination NSB client. |
| <i>payload</i> | The payload to post to the destination. |
| <i>success</i> | Whether the post was successful or not. If False, it will set the OpCode to NO_MESSAGE. |

6.6.4 Member Data Documentation

logger

```
nsb_client.NSBSimClient.logger = logging.getLogger(" (SimClient) ")
```

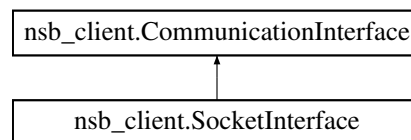
The documentation for this class was generated from the following file:

- [nsb_client.py](#)

6.7 nsb_client.SocketInterface Class Reference

Socket interface for client-server communication.

Inheritance diagram for nsb_client.SocketInterface:



Public Member Functions

- `__init__` (self, str server_address, int [server_port](#))
Constructor for the [NSBClient](#) class.
- `__del__` (self)
Closes connection to the server.

Public Attributes

- [server_addr](#) = server_address
- [server_port](#) = server_port
- [logger](#) = logging.getLogger("NSBClient")
- [conn](#) = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

Protected Member Functions

- [_configure](#) (self)
Configures the socket connection with appropriate options.
- [_connect](#) (self, int timeout=[SERVER_CONNECTION_TIMEOUT](#))
Connects to the daemon with the stored server address and port.
- [_close](#) (self)
Healthily closes the socket connection.
- [_send_msg](#) (self, bytes message)
Sends a message to the server.
- [_recv_msg](#) (self, int|None timeout=None)
Sends a message to the server.

6.7.1 Detailed Description

Socket interface for client-server communication.

This class implements aocket interface network to facilitate network communication between NSB clients and the server. This can be used as a template to develop other interfaces for client communication, which must define the same methods with the same arguments as done in this class.

6.7.2 Constructor & Destructor Documentation

`__init__()`

```
nsb_client.SocketInterface.__init__ (  
    self,  
    str server_address,  
    int server_port)
```

Constructor for the [NSBClient](#) class.

Sets the address and port of the server at the NSB daemon before connecting to the server.

Parameters

| | |
|-----------------------|--------------------------------|
| <i>server_address</i> | The address of the NSB daemon. |
| <i>server_port</i> | The port of the NSB daemon. |

See also

`nsb_client.SocketInterface._connect(timeout)`

`__del__()`

```
nsb_client.SocketInterface.__del__ (  
    self)
```

Closes connection to the server.

See also

[_close\(\)](#)

6.7.3 Member Function Documentation

`_close()`

```
nsb_client.SocketInterface._close (  
    self) [protected]
```

Healthily closes the socket connection.

Attempts to shutdown the socket, then closes.

`_configure()`

```
nsb_client.SocketInterface._configure (  
    self) [protected]
```

Configures the socket connection with appropriate options.

This method is called by the [_connect\(\)](#) method. It configures the sockets to work with the multi-connection server at the daemon with lower latency.

See also

[_connect\(\)](#)

`_connect()`

```
nsb_client.SocketInterface._connect (  
    self,  
    int timeout = SERVER\_CONNECTION\_TIMEOUT) [protected]
```

Connects to the daemon with the stored server address and port.

This method uses the [_configure\(\)](#) method to configure the socket and then attempts to connect to the daemon.

Parameters

| | |
|----------------|---|
| <i>timeout</i> | Maximum time in seconds to wait to connect to the daemon. |
|----------------|---|

Exceptions

| | |
|---------------------|--|
| <i>TimeoutError</i> | Raised if the connection to the server times out after the specified timeout period. |
|---------------------|--|

`_recv_msg()`

```
nsb_client.SocketInterface._recv_msg (  
    self,  
    int|None timeout = None) [protected]
```

Sends a message to the server.

This method uses selectors to wait for the socket to be ready before sending SEND_BUFFER SIZE bytes at a time, making it non-blocking compliant.

Parameters

| | |
|----------------|---|
| <i>timeout</i> | Maximum time in seconds to wait for a response from the server. If None, it will wait indefinitely. |
|----------------|---|

`_send_msg()`

```
nsb_client.SocketInterface._send_msg (  
    self,  
    bytes message) [protected]
```

Sends a message to the server.

This method uses selectors to wait for the socket to be ready before sending SEND_BUFFER SIZE bytes at a time, making it non-blocking compliant.

Parameters

| | |
|----------------|------------------------------------|
| <i>message</i> | The message to send to the server. |
|----------------|------------------------------------|

Exceptions

| | |
|---------------------|--|
| <i>RuntimeError</i> | Raised if the socket connection is broken or if the socket is not ready to send. |
|---------------------|--|

6.7.4 Member Data Documentation

conn

```
nsb_client.SocketInterface.conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

logger

```
nsb_client.SocketInterface.logger = logging.getLogger("NSBClient")
```

server_addr

```
nsb_client.SocketInterface.server_addr = server_address
```

server_port

```
nsb_client.SocketInterface.server_port = server_port
```

The documentation for this class was generated from the following file:

- [nsb_client.py](#)

7 File Documentation

7.1 nsb_client.py File Reference

Classes

- class [nsb_client.CommunicationInterface](#)
Base class for communication interfaces.
- class [nsb_client.SocketInterface](#)
Socket interface for client-server communication.
- class [nsb_client.NSBClient](#)
NSB client base class.
- class [nsb_client.NSBAppClient](#)
NSB Application Client interface.
- class [nsb_client.NSBSimClient](#)
NSB Simulator Client interface.

Namespaces

- namespace `nsb_client`

Functions

- `nsb_client.test_ping ()`
- `nsb_client.test_lifecycle ()`

Variables

- `int nsb_client.SERVER_CONNECTION_TIMEOUT = 10`
- `int nsb_client.DAEMON_RESPONSE_TIMEOUT = 600`
- `int nsb_client.RECEIVE_BUFFER_SIZE = 4096`
Buffer size when receiving data.
- `int nsb_client.SEND_BUFFER_SIZE = 4096`
Buffer size when sending data.

7.2 nsb_daemon.cc File Reference

```
#include "nsb_daemon.h"
```

Functions

- `int main ()`
Main process to run the NSB Daemon.

7.2.1 Function Documentation

`main()`

```
int main ()
```

Main process to run the NSB Daemon.

Returns

`int`

7.3 nsb_daemon.h File Reference

```
#include <string>
#include <list>
#include <vector>
#include <map>
#include <array>
#include <atomic>
#include <thread>
#include <iostream>
#include <cstdio>
#include <format>
#include <signal.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <fcntl.h>
#include <sqlite3.h>
#include "nsb.pb.h"
```

Classes

- struct [MessageEntry](#)
Message storage struct.
- class [NSBDaemon](#)

Variables

- int [MAX_BUFFER_SIZE](#) = 4096
The maximum buffer size for sending and receiving messages.

7.3.1 Variable Documentation

MAX_BUFFER_SIZE

```
int MAX_BUFFER_SIZE = 4096
```

The maximum buffer size for sending and receiving messages.

7.4 nsb_daemon.h

[Go to the documentation of this file.](#)

```

00001 // nsb_daemon.h
00002
00003 #ifndef NSB_DAEMON_H
00004 #define NSB_DAEMON_H
00005
00006 #include <string>
00007 #include <list>
00008 #include <vector>
00009 #include <map>
00010 #include <array>
00011 // Thread libraries.
00012 #include <atomic>
00013 #include <thread>
00014 // I/O libraries.
00015 #include <iostream>
00016 #include <cstdio>
00017 #include <format>
00018 #include <signal.h>
00019 // Networking libraries.
00020 #include <arpa/inet.h>
00021 #include <netinet/in.h>
00022 #include <sys/socket.h>
00023 #include <unistd.h>
00024 #include <fcntl.h>
00025 // Data.
00026 #include <sqlite3.h>
00027
00028 #include "nsb.pb.h"
00029
00030
00031 int MAX_BUFFER_SIZE = 4096;
00032
00033 struct MessageEntry {
00034     std::string source;
00035     std::string destination;
00036     std::string payload;
00037     // Constructors.
00038     MessageEntry() : source(""), destination(""), payload("") {}
00039     MessageEntry(std::string src, std::string dest, std::string data)
00040         : source(std::move(src)), destination(std::move(dest)), payload(std::move(data)) {}
00041 };
00042
00043 class NSBDaemon {
00044 public:
00045     NSBDaemon(int s_port);
00046     ~NSBDaemon();
00047     void start();
00048     void stop();
00049     bool is_running() const;
00050 private:
00051     std::atomic<bool> running;
00052     int server_port;
00053     std::list<MessageEntry> tx_buffer;
00054     std::list<MessageEntry> rx_buffer;
00055     void start_server(int port);
00056     void handle_message(int fd, std::vector<char> message);
00057
00058     /* Operation-specific handlers. */
00059
00060     void handle_ping(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00061     void handle_send(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00062     void handle_fetch(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00063     void handle_post(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00064     void handle_receive(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00065 };
00066
00067 #endif // NSB_DAEMON_H

```

Index

- `__del__`
 - `nsb_client.SocketInterface`, 21
 - `__init__`
 - `nsb_client.NSBApClient`, 7
 - `nsb_client.NSBClient`, 9
 - `nsb_client.NSBSimClient`, 18
 - `nsb_client.SocketInterface`, 21
 - `_close`
 - `nsb_client.SocketInterface`, 21
 - `_configure`
 - `nsb_client.SocketInterface`, 21
 - `_connect`
 - `nsb_client.SocketInterface`, 22
 - `_id`
 - `nsb_client.NSBApClient`, 8
 - `_recv_msg`
 - `nsb_client.SocketInterface`, 22
 - `_send_msg`
 - `nsb_client.SocketInterface`, 22
 - `~NSBDaemon`
 - `NSBDaemon`, 11
- `comms`
 - `nsb_client.NSBClient`, 10
- `conn`
 - `nsb_client.SocketInterface`, 23
- `DAEMON_RESPONSE_TIMEOUT`
 - `nsb_client`, 3
- `destination`
 - `MessageEntry`, 5
- `exit`
 - `nsb_client.NSBClient`, 9
- `fetch`
 - `nsb_client.NSBSimClient`, 19
- `handle_fetch`
 - `NSBDaemon`, 12
- `handle_message`
 - `NSBDaemon`, 12
- `handle_ping`
 - `NSBDaemon`, 12
- `handle_post`
 - `NSBDaemon`, 13
- `handle_receive`
 - `NSBDaemon`, 13
- `handle_send`
 - `NSBDaemon`, 15
- `is_running`
 - `NSBDaemon`, 15
- `logger`
 - `nsb_client.NSBApClient`, 8
 - `nsb_client.NSBSimClient`, 20

- `nsb_client.SocketInterface`, 23
- `main`
 - `nsb_daemon.cc`, 24
- `MAX_BUFFER_SIZE`
 - `nsb_daemon.h`, 25
- `MessageEntry`, 4
 - `destination`, 5
 - `MessageEntry`, 5
 - `payload`, 5
 - `source`, 5
- `nsb_client`, 2
 - `DAEMON_RESPONSE_TIMEOUT`, 3
 - `RECEIVE_BUFFER_SIZE`, 3
 - `SEND_BUFFER_SIZE`, 3
 - `SERVER_CONNECTION_TIMEOUT`, 3
 - `test_lifecycle`, 3
 - `test_ping`, 3
- `nsb_client.CommunicationInterface`, 4
- `nsb_client.NSBApClient`, 6
 - `__init__`, 7
 - `_id`, 8
 - `logger`, 8
 - `receive`, 7
 - `send`, 8
- `nsb_client.NSBClient`, 8
 - `__init__`, 9
 - `comms`, 10
 - `exit`, 9
 - `ping`, 9
- `nsb_client.NSBSimClient`, 17
 - `__init__`, 18
 - `fetch`, 19
 - `logger`, 20
 - `post`, 19
- `nsb_client.py`, 23
- `nsb_client.SocketInterface`, 20
 - `__del__`, 21
 - `__init__`, 21
 - `_close`, 21
 - `_configure`, 21
 - `_connect`, 22
 - `_recv_msg`, 22
 - `_send_msg`, 22
 - `conn`, 23
 - `logger`, 23
 - `server_addr`, 23
 - `server_port`, 23
- `nsb_daemon.cc`, 24
 - `main`, 24
- `nsb_daemon.h`, 25
 - `MAX_BUFFER_SIZE`, 25
- `NSBDaemon`, 10
 - `~NSBDaemon`, 11
 - `handle_fetch`, 12

- handle_message, 12
- handle_ping, 12
- handle_post, 13
- handle_receive, 13
- handle_send, 15
- is_running, 15
- NSBDaemon, 11
- running, 16
- rx_buffer, 16
- server_port, 17
- start, 15
- start_server, 16
- stop, 16
- tx_buffer, 17

payload

- MessageEntry, 5

ping

- nsb_client.NSBClient, 9

post

- nsb_client.NSBSimClient, 19

receive

- nsb_client.NSBAppClient, 7

RECEIVE_BUFFER_SIZE

- nsb_client, 3

running

- NSBDaemon, 16

rx_buffer

- NSBDaemon, 16

send

- nsb_client.NSBAppClient, 8

SEND_BUFFER_SIZE

- nsb_client, 3

server_addr

- nsb_client.SocketInterface, 23

SERVER_CONNECTION_TIMEOUT

- nsb_client, 3

server_port

- nsb_client.SocketInterface, 23
- NSBDaemon, 17

source

- MessageEntry, 5

start

- NSBDaemon, 15

start_server

- NSBDaemon, 16

stop

- NSBDaemon, 16

test_lifecycle

- nsb_client, 3

test_ping

- nsb_client, 3

tx_buffer

- NSBDaemon, 17