NSB

0.1.0

Generated by Doxygen 1.14.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 MessageEntry Struct Reference

Message storage struct.

`#include <nsb_daemon.h>`

Public Member Functions

- MessageEntry ()
- MessageEntry (std::string src, std::string dest, std::string data)

Public Attributes

- std::string source
- std::string destination
- std::string payload

3.1.1 Detailed Description

Message storage struct.

Contains source and destination information and the payload.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 MessageEntry() [1/2]

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

3.1.2.2 MessageEntry() [2/2]

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

3.1.3 Member Data Documentation

3.1.3.1 destination

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

3.1.3.2 payload

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

3.1.3.3 source

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

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

- nsb_daemon.h

3.2 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
- int server_port
- std::list< MessageEntry > tx_buffer
- std::list< MessageEntry > rx_buffer

3.2.1 Constructor & Destructor Documentation

3.2.1.1 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.

3.2.1.2 ~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.

3.2.2 Member Function Documentation

3.2.2.1 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

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

See also

MessageEntry

3.2.2.2 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

fd	The file descriptor of the client connection.
message	The incoming message to parse and handle.

See also

NSBDaemon::start_server(int port)

NSBDaemon::handle_ping(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

NSBDaemon::handle_send(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

NSBDaemon::handle_fetch(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

NSBDaemon::handle_post(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

NSBDaemon::handle_receive(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

3.2.2.3 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

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

3.2.2.4 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

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

See also

MessageEntry

NSBDaemon::handle_receive(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

3.2.2.5 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

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

See also

MessageEntry

3.2.2.6 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

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

See also

MessageEntry

NSBDaemon::handle_fetch(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required)

3.2.2.7 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.

3.2.2.8 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

NSBDaemon::start_server(int port)

3.2.2.9 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

port	The port that will be accessible for clients to connect.

See also

NSBDaemon::start()

NSBDaemon::handle_message(int fd, std::vector<char> message)

3.2.2.10 stop()

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

Stops the NSB Daemon.

3.2.3 Member Data Documentation

3.2.3.1 running

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

3.2.3.2 rx_buffer

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

3.2.3.3 server_port

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

3.2.3.4 tx_buffer

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

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

- nsb_daemon.h
- nsb_daemon.cc

Chapter 4

File Documentation

4.1 nsb_daemon.cc File Reference

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

Functions

- int main ()

  Main process to run the NSB Daemon.

4.1.1 Function Documentation

4.1.1.1 main()

```
int main ()
```

Main process to run the NSB Daemon.

Returns

int

4.2 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

4.2.1 Variable Documentation

4.2.1.1 MAX_BUFFER_SIZE

```
int MAX_BUFFER_SIZE = 4096
```

4.3 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 // Decide whether to use threads per connection (not recommended for now).
00031 // #define NSB_USE_THREADS
00032 // Decide whether to use database.
00033 // #define NSB_USE_DB
00034
00035 int MAX_BUFFER_SIZE = 4096;
00036
00043 struct MessageEntry {
00044     std::string source;
00045     std::string destination;
00046     std::string payload;
00047     // Constructors.
00048     MessageEntry() : source(""), destination(""), payload("") {}
00049     MessageEntry(std::string src, std::string dest, std::string data)
00050         : source(std::move(src)), destination(std::move(dest)), payload(std::move(data)) {}
00051 };
00052
00053 class NSBDaemon {
00054 public:
00055     NSBDaemon(int s_port);
00056     ~NSBDaemon();
00057     void start();
00058     void stop();
00059     bool is_running() const;
00060
00061 private:
00062     std::atomic<bool> running;
00063     int server_port;
00064     std::list<MessageEntry> tx_buffer;
00065     std::list<MessageEntry> rx_buffer;
00066     void start_server(int port);
00067     void handle_message(int fd, std::vector<char> message);
00068     // Operation-specific handlers.
00069     void handle_ping(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00070     void handle_send(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00071     void handle_fetch(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00072     void handle_post(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00073     void handle_receive(nsb::nsbm* incoming_msg, nsb::nsbm* outgoing_msg, bool* response_required);
00074 };
00075
00076 #endif // NSB_DAEMON_H
```

Index