High-Performance order execution and management system

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 deriapi Namespace Reference

Functions

- std::string authorize (const std::string &clientId, const std::string &clientSecret)

  Authorizes the client using client ID and secret.
- std::string getAccountSummary (const std::string &currency)

  Retrieves the account summary for a specific currency.
- std::string createOrder (const std::string &method, const std::string &instrument, int amount, const std::string &orderType, double price, const std::string &timeInForce, const std::string &label, const std::string &accessToken, bool postOnly=false)

  Creates an order request (buy or sell) with the specified parameters.
- std::string buyOrder (const std::string &instrument, int amount, const std::string &orderType, int price, const std::string &timeInForce, const std::string &label, const std::string &accessToken)

  Creates a buy order request.
- std::string sellOrder (const std::string &instrument, int amount, const std::string &orderType, double price, const std::string &timeInForce, const std::string &label, const std::string &accessToken, bool postOnly)

  Creates a sell order request.
- std::string cancelOrder (const std::string &orderId)

  Creates a cancel order request.
- std::string getOrderBook (const std::string &instrumentName, int depth)

  Creates a request to retrieve the order book for a specific instrument.
- std::string modifyOrder (const std::string &orderId, int amount, double price, const std::string &timeInForce, bool postOnly, bool reduceOnly)

  Creates a request to modify an existing order.
- std::string getPositions (const std::string &currency, const std::string &kind)

  Creates a request to retrieve positions for a specific currency and kind.
- std::string subscribeToChannel (const std::string &channel)

  Creates a request to subscribe to a WebSocket channel.
- std::string unsubscribeFromChannel (const std::string &channel)

  Creates a request to unsubscribe from a WebSocket channel.

4.1.1 Function Documentation

4.1.1.1 authorize()

```
std::string deriapi::authorize (
            const std::string & clientId,
            const std::string & clientSecret )
```

Authorizes the client using client ID and secret.

This function generates a timestamp, nonce, and client signature, then creates an authorization request JSON object using the "client_signature" grant type.

Parameters

clientId	The client ID for authentication.
clientSecret	The client secret for authentication.

Returns

std::string The authorization request in JSON format.

Exceptions

std::runtime_error	If the client signature cannot be generated.

Definition at line 30 of file deriapi.cpp.

4.1.1.2 buyOrder()

```
std::string deriapi::buyOrder (
            const std::string & instrument,
            int amount,
            const std::string & orderType,
            int price,
            const std::string & timeInForce,
            const std::string & label,
            const std::string & accessToken )
```

Creates a buy order request.

This function generates a JSON request for placing a buy order using the `createOrder` function.

Parameters

instrument	The instrument name (e.g., "BTC-PERPETUAL").
amount	The amount of the instrument to buy.
orderType	The type of order (e.g., "limit", "market", "stop_limit").
price	The price for limit or stop-limit orders.
timeInForce	The time-in-force for the order (e.g., "good_til_cancelled").
label	A custom label for the order.
accessToken	The access token for authentication.

Returns

std::string The buy order request in JSON format.

This function generates a JSON request for placing a buy order with the specified parameters.

Parameters

instrument	The instrument name (e.g., "BTC-PERPETUAL").
amount	The amount of the instrument to buy.
orderType	The type of order (e.g., "limit", "market", "stop_limit").
price	The price for limit or stop-limit orders.
timeInForce	The time-in-force for the order (e.g., "good_til_cancelled").
label	A custom label for the order.
accessToken	The access token for authentication.

Returns

std::string The buy order request in JSON format.

Definition at line 124 of file deriapi.cpp.

4.1.1.3 cancelOrder()

```
std::string deriapi::cancelOrder (
            const std::string & orderId )
```

Creates a cancel order request.

This function generates a JSON request for canceling an order using the specified order ID.

Parameters

orderId	The ID of the order to cancel.

Returns

std::string The cancel order request in JSON format.

Definition at line 155 of file deriapi.cpp.

4.1.1.4 createOrder()

```
std::string deriapi::createOrder (
            const std::string & method,
            const std::string & instrument,
            int amount,
            const std::string & orderType,
```

```
            double price,
            const std::string & timeInForce,
            const std::string & label,
            const std::string & accessToken,
            bool postOnly = false )
```

Creates an order request (buy or sell) with the specified parameters.

This function generates a JSON request for placing an order (buy or sell) with the given parameters. It supports limit, market, and stop-limit order types.

Parameters

method	The order method (e.g., "private/buy" or "private/sell").
instrument	The instrument name (e.g., "BTC-PERPETUAL").
amount	The amount of the instrument to buy or sell.
orderType	The type of order (e.g., "limit", "market", "stop_limit").
price	The price for limit or stop-limit orders.
timeInForce	The time-in-force for the order (e.g., "good_til_cancelled").
label	A custom label for the order.
accessToken	The access token for authentication.
postOnly	[optional] Whether the order should be post-only (default: false).

Returns

std::string The order request in JSON format.

Definition at line 89 of file deriapi.cpp.

4.1.1.5 getAccountSummary()

```
std::string deriapi::getAccountSummary (
            const std::string & currency )
```

Retrieves the account summary for a specific currency.

This function creates a JSON request to get the account summary for the specified currency.

Parameters

currency	The currency for which to retrieve the account summary (e.g., "BTC").

Returns

std::string The account summary request in JSON format.

Definition at line 60 of file deriapi.cpp.

4.1.1.6 getOrderBook()

```
std::string deriapi::getOrderBook (
            const std::string & instrumentName,
            int depth )
```

Creates a request to retrieve the order book for a specific instrument.

This function generates a JSON request to get the order book for the specified instrument and depth.

Parameters

instrumentName	The name of the instrument (e.g., "BTC-PERPETUAL").
depth	The depth of the order book to retrieve.

Returns

std::string The order book request in JSON format.

Definition at line 176 of file deriapi.cpp.

4.1.1.7 getPositions()

```
std::string deriapi::getPositions (
            const std::string & currency,
            const std::string & kind )
```

Creates a request to retrieve positions for a specific currency and kind.

This function generates a JSON request to get positions for the specified currency and kind.

Parameters

currency	The currency for which to retrieve positions (e.g., "BTC").
kind	The kind of positions to retrieve (e.g., "future", "option").

Returns

std::string The positions request in JSON format.

This function generates a JSON request to get positions for the specified currency and kind.

Parameters

currency	The currency for which to retrieve positions (e.g., "BTC").
kind	[optional] The kind of positions to retrieve (e.g., "future", "option"). Default: "future".

Returns

std::string The positions request in JSON format.

Definition at line 228 of file deriapi.cpp.

4.1.1.8 modifyOrder()

```
std::string deriapi::modifyOrder (
            const std::string & orderId,
            int amount,
            double price,
            const std::string & timeInForce,
            bool postOnly,
            bool reduceOnly )
```

Creates a request to modify an existing order.

This function generates a JSON request to modify an order with the specified parameters.

Parameters

orderId	The ID of the order to modify.
amount	The new amount for the order.
price	The new price for the order.
timeInForce	The new time-in-force for the order.
postOnly	Whether the order should be post-only.
reduceOnly	Whether the order should be reduce-only.

Returns

std::string The modify order request in JSON format.

This function generates a JSON request to modify an order with the specified parameters.

Parameters

orderId	The ID of the order to modify.
amount	The new amount for the order.
price	The new price for the order.
timeInForce	The new time-in-force for the order.
postOnly	[optional] Whether the order should be post-only (default: false).
reduceOnly	[optional] Whether the order should be reduce-only (default: false).

Returns

std::string The modify order request in JSON format.

Definition at line 202 of file deriapi.cpp.

4.1.1.9 sellOrder()

```
std::string deriapi::sellOrder (
            const std::string & instrument,
            int amount,
            const std::string & orderType,
            double price,
            const std::string & timeInForce,
            const std::string & label,
            const std::string & accessToken,
            bool postOnly )
```

Creates a sell order request.

This function generates a JSON request for placing a sell order using the `createOrder` function.

Parameters

instrument	The instrument name (e.g., "BTC-PERPETUAL").
amount	The amount of the instrument to sell.
orderType	The type of order (e.g., "limit", "market", "stop_limit").
price	The price for limit or stop-limit orders.
timeInForce	The time-in-force for the order (e.g., "good_til_cancelled").
label	A custom label for the order.
accessToken	The access token for authentication.
postOnly	[optional] Whether the order should be post-only (default: false).

Returns

std::string The sell order request in JSON format.

This function generates a JSON request for placing a sell order with the specified parameters.

Parameters

instrument	The instrument name (e.g., "BTC-PERPETUAL").
amount	The amount of the instrument to sell.
orderType	The type of order (e.g., "limit", "market", "stop_limit").
price	The price for limit or stop-limit orders.
timeInForce	The time-in-force for the order (e.g., "good_til_cancelled").
label	A custom label for the order.
accessToken	The access token for authentication.
postOnly	[optional] Whether the order should be post-only (default: false).

Returns

std::string The sell order request in JSON format.

Definition at line 143 of file deriapi.cpp.

4.1.1.10 subscribeToChannel()

```
std::string deriapi::subscribeToChannel (
            const std::string & channel )
```

Creates a request to subscribe to a WebSocket channel.

This function generates a JSON request to subscribe to the specified WebSocket channel.

Parameters

channel	The channel to subscribe to (e.g., "ticker.BTC-PERPETUAL.100ms").

Returns

std::string The subscription request in JSON format.

Definition at line 249 of file deriapi.cpp.

4.1.1.11 unsubscribeFromChannel()

```
std::string deriapi::unsubscribeFromChannel (
            const std::string & channel )
```

Creates a request to unsubscribe from a WebSocket channel.

This function generates a JSON request to unsubscribe from the specified WebSocket channel.

Parameters

channel	The channel to unsubscribe from (e.g., "ticker.BTC-PERPETUAL.100ms").

Returns

std::string The unsubscription request in JSON format.

Definition at line 269 of file deriapi.cpp.

4.2 utils Namespace Reference

Functions

- std::string getTimeStamp ()

  Generates a timestamp in milliseconds since the Unix epoch.
- std::string getNonce ()

  Generates a random nonce of 8 characters.
- std::string toHex (const unsigned char *data, size_t length)

  Converts binary data to a hexadecimal string.

- std::string hmacSha256 (const std::string &secret, const std::string &data)

  Computes the HMAC-SHA256 hash of the given data using the provided secret.
- std::string getClientSignature (const std::string &clientSecret, const std::string &timeStamp, const std::string &nonce, const std::string &data)

  Generates a client signature using the provided client secret, timestamp, nonce, and data.

4.2.1 Function Documentation

4.2.1.1 getClientSignature()

```
std::string utils::getClientSignature (
            const std::string & clientSecret,
            const std::string & timeStamp,
            const std::string & nonce,
            const std::string & data )
```

Generates a client signature using the provided client secret, timestamp, nonce, and data.

This function creates a string to sign by concatenating the timestamp, nonce, and data, then computes the HMAC-SHA256 hash of the string using the client secret.

Parameters

clientSecret	The client secret for the HMAC computation.
timeStamp	The timestamp to include in the signature.
nonce	The nonce to include in the signature.
data	Additional data to include in the signature (optional).

Returns

std::string The client signature as a hexadecimal string.

This function creates a string to sign by concatenating the timestamp, nonce, and data, then computes the HMAC-SHA256 hash of the string using the client secret.

Parameters

clientSecret	The client secret for the HMAC computation.
timeStamp	The timestamp to include in the signature.
nonce	The nonce to include in the signature.
data	[optional] Additional data to include in the signature.

Returns

std::string The client signature as a hexadecimal string.

Definition at line 103 of file utils.cpp.

4.2.1.2 getNonce()

```
std::string utils::getNonce ( )
```

Generates a random nonce of 8 characters.

This function creates a random string of 8 characters using alphanumeric characters.

Returns

std::string The generated nonce.

Definition at line 40 of file utils.cpp.

4.2.1.3 getTimeStamp()

```
std::string utils::getTimeStamp ( )
```

Generates a timestamp in milliseconds since the Unix epoch.

This function retrieves the current system time and converts it to milliseconds.

Returns

std::string The timestamp as a string.

Definition at line 27 of file utils.cpp.

4.2.1.4 hmacSha256()

```
std::string utils::hmacSha256 (
            const std::string & secret,
            const std::string & data )
```

Computes the HMAC-SHA256 hash of the given data using the provided secret.

This function uses OpenSSL's HMAC function to compute the HMAC-SHA256 hash.

Parameters

secret	The secret key for the HMAC computation.
data	The data to hash.

Returns

std::string The HMAC-SHA256 hash as a hexadecimal string.

Definition at line 80 of file utils.cpp.

4.2.1.5 toHex()

```
std::string utils::toHex (
            const unsigned char * data,
            size_t length )
```

Converts binary data to a hexadecimal string.

This function takes binary data and converts it to a hexadecimal representation.

Parameters

data	The binary data to convert.
length	The length of the binary data.

Returns

std::string The hexadecimal representation of the data.

Definition at line 62 of file utils.cpp.

Chapter 5

Class Documentation

5.1 webSocketClient Class Reference

A WebSocket client for interacting with a WebSocket server.

`#include <webSocketClient.h>`

Public Member Functions

- webSocketClient ()
  Constructs a new WebSocket client.
- ~webSocketClient ()
  Destructor for the WebSocket client.
- void setAuthRequestCallback (std::function< void()> callback)
  Sets the authentication request callback.
- void send (const std::string &message)
  Sends a message through the WebSocket connection.
- void connect (const std::string &uri)
  Connects to a WebSocket server.
- void close ()
  Closes the WebSocket connection.
- void subscribe (const std::string &channel)
  Subscribes to a WebSocket channel.
- void unsubscribe (const std::string &channel)
  Unsubscribes from a WebSocket channel.
- bool isAuthenticated () const
  Checks if the client is authenticated.
- bool isWaitingForResponse () const
  Checks if the client is waiting for a response.
- std::string getAccessToken () const
  Gets the access token.

Private Member Functions

- void on_open (client *c, websocketpp::connection_hdl hdl)

  Handles the WebSocket connection open event.
- void on_fail (client *c, websocketpp::connection_hdl hdl)

  Handles the WebSocket connection fail event.
- void on_close (client *c, websocketpp::connection_hdl hdl)

  Handles the WebSocket connection close event.
- void on_message (client *c, websocketpp::connection_hdl hdl, client::message_ptr msg)

  Handles incoming WebSocket messages.
- void handleSubscriptionMessage (const std::string &channel, const nlohmann::json &data)

  Handles subscription messages for a specific channel.
- void on_message_auth (nlohmann::json result)

  Handles authentication success messages.
- void on_message_summary (nlohmann::json result)

  Handles account summary messages.
- void on_message_buy (nlohmann::json order)

  Handles buy order success messages.
- void on_message_cancel (nlohmann::json result)

  Handles order cancellation success messages.
- void on_message_orderBook (nlohmann::json result)

  Handles order book update messages.
- void on_message_modify (nlohmann::json result)

  Handles order modification success messages.
- void on_message_positions (nlohmann::json result)

  Handles position update messages.
- void on_message_sell (nlohmann::json result)

  Handles sell order success messages.

Private Attributes

- client m_endpoint

  The WebSocket endpoint.
- websocketpp::connection_hdl m_hdl

  The connection handle.
- std::thread m_eventLoopThread

  The thread running the WebSocket event loop.
- bool m_connected

  Indicates whether the client is connected to the server.
- std::function< void()> m_authRequestCallback

  Callback function for authentication requests.
- bool m_authenticated

  Indicates whether the client is authenticated.
- bool m_waitingForResponse

  Indicates whether the client is waiting for a response.
- std::string m_accessToken

  The access token for authenticated sessions.
- std::map< std::string, std::string > m_lastData

  Stores the last received data for each channel.
- std::set< std::string > m_subscribedChannels

  Stores the names of subscribed channels.

5.1.1 Detailed Description

A WebSocket client for interacting with a WebSocket server.

This class provides methods to connect to a WebSocket server, send and receive messages, and handle events such as connection open, close, and message reception. It also supports subscription to channels and authentication.

Definition at line 41 of file webSocketClient.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 webSocketClient()

```
webSocketClient::webSocketClient ( )
```

Constructs a new WebSocket client.

Initializes the WebSocket endpoint, disables logging, sets up TLS, and configures event handlers.

Definition at line 19 of file webSocketClient.cpp.

5.1.2.2 ~webSocketClient()

```
webSocketClient::~webSocketClient ( )
```

Destructor for the WebSocket client.

Ensures the WebSocket connection is properly closed and resources are cleaned up.

Stops the perpetual loop and closes the connection if it is still open.

Definition at line 49 of file webSocketClient.cpp.

5.1.3 Member Function Documentation

5.1.3.1 close()

```
void webSocketClient::close ( )
```

Closes the WebSocket connection.

Definition at line 98 of file webSocketClient.cpp.

5.1.3.2 connect()

```
void webSocketClient::connect (
            const std::string & uri )
```

Connects to a WebSocket server.

Parameters

uri	The URI of the WebSocket server to connect to.

Definition at line 83 of file webSocketClient.cpp.

5.1.3.3 getAccessToken()

```
std::string webSocketClient::getAccessToken ( ) const
```

Gets the access token.

Returns

The access token as a string.

Definition at line 498 of file webSocketClient.cpp.

5.1.3.4 handleSubscriptionMessage()

```
void webSocketClient::handleSubscriptionMessage (
            const std::string & channel,
            const nlohmann::json & data )  [private]
```

Handles subscription messages for a specific channel.

Parameters

channel	The name of the channel.
data	The JSON data received for the channel.

Definition at line 185 of file webSocketClient.cpp.

5.1.3.5 isAuthenticated()

```
bool webSocketClient::isAuthenticated ( ) const
```

Checks if the client is authenticated.

Returns

True if authenticated, false otherwise.

Definition at line 480 of file webSocketClient.cpp.

5.1.3.6 isWaitingForResponse()

```
bool webSocketClient::isWaitingForResponse ( ) const
```

Checks if the client is waiting for a response.

Returns

True if waiting for a response, false otherwise.

Definition at line 489 of file webSocketClient.cpp.

5.1.3.7 on_close()

```
void webSocketClient::on_close (
            client * c,
            websocketpp::connection_hdl hdl )  [private]
```

Handles the WebSocket connection close event.

Parameters

c	Pointer to the WebSocket client.
hdl	The connection handle.

Definition at line 148 of file webSocketClient.cpp.

5.1.3.8 on_fail()

```
void webSocketClient::on_fail (
            client * c,
            websocketpp::connection_hdl hdl )  [private]
```

Handles the WebSocket connection fail event.

Parameters

c	Pointer to the WebSocket client.
hdl	The connection handle.

Definition at line 137 of file webSocketClient.cpp.

5.1.3.9 on_message()

```
void webSocketClient::on_message (
            client * c,
            websocketpp::connection_hdl hdl,
            client::message_ptr msg )  [private]
```

Handles incoming WebSocket messages.

Parameters

c	Pointer to the WebSocket client.
hdl	The connection handle.
msg	The received message.

Definition at line 395 of file webSocketClient.cpp.

5.1.3.10 on_message_auth()

```
void webSocketClient::on_message_auth (
            nlohmann::json result )  [private]
```

Handles authentication success messages.

Parameters

result	The JSON result containing authentication details.

Definition at line 224 of file webSocketClient.cpp.

5.1.3.11 on_message_buy()

```
void webSocketClient::on_message_buy (
            nlohmann::json order )  [private]
```

Handles buy order success messages.

Parameters

order	The JSON result containing buy order details.

Definition at line 250 of file webSocketClient.cpp.

5.1.3.12 on_message_cancel()

```
void webSocketClient::on_message_cancel (
            nlohmann::json result )  [private]
```

Handles order cancellation success messages.

Parameters

result	The JSON result containing cancellation details.

Definition at line 270 of file webSocketClient.cpp.

5.1.3.13 on_message_modify()

```
void webSocketClient::on_message_modify (
            nlohmann::json result )  [private]
```

Handles order modification success messages.

Parameters

result	The JSON result containing modification details.

Definition at line 332 of file webSocketClient.cpp.

5.1.3.14 on_message_orderBook()

```
void webSocketClient::on_message_orderBook (
            nlohmann::json result )  [private]
```

Handles order book update messages.

Parameters

result	The JSON result containing order book details.

Definition at line 282 of file webSocketClient.cpp.

5.1.3.15 on_message_positions()

```
void webSocketClient::on_message_positions (
            nlohmann::json result )  [private]
```

Handles position update messages.

Parameters

result	The JSON result containing position details.

Definition at line 345 of file webSocketClient.cpp.

5.1.3.16 on_message_sell()

```
void webSocketClient::on_message_sell (
            nlohmann::json result )  [private]
```

Handles sell order success messages.

Parameters

result	The JSON result containing sell order details.

Definition at line 373 of file webSocketClient.cpp.

5.1.3.17 on_message_summary()

```
void webSocketClient::on_message_summary (
            nlohmann::json result )  [private]
```

Handles account summary messages.

Parameters

result	The JSON result containing account summary details.

Definition at line 233 of file webSocketClient.cpp.

5.1.3.18 on_open()

```
void webSocketClient::on_open (
            client * c,
            websocketpp::connection_hdl hdl )  [private]
```

Handles the WebSocket connection open event.

Parameters

c	Pointer to the WebSocket client.
hdl	The connection handle.

Definition at line 122 of file webSocketClient.cpp.

5.1.3.19 send()

```
void webSocketClient::send (
            const std::string & message )
```

Sends a message through the WebSocket connection.

Parameters

message	The message to send.

Definition at line 70 of file webSocketClient.cpp.

5.1.3.20 setAuthRequestCallback()

```
void webSocketClient::setAuthRequestCallback (
            std::function< void()> callback )
```

Sets the authentication request callback.

Parameters

callback	A function to be called when authentication is requested.

Definition at line 61 of file webSocketClient.cpp.

5.1.3.21 subscribe()

```
void webSocketClient::subscribe (
            const std::string & channel )
```

Subscribes to a WebSocket channel.

Parameters

channel	The name of the channel to subscribe to.

Definition at line 158 of file webSocketClient.cpp.

5.1.3.22 unsubscribe()

```
void webSocketClient::unsubscribe (
            const std::string & channel )
```

Unsubscribes from a WebSocket channel.

Parameters

channel	The name of the channel to unsubscribe from.

Definition at line 170 of file webSocketClient.cpp.

5.1.4 Member Data Documentation

5.1.4.1 m_accessToken

```
std::string webSocketClient::m_accessToken  [private]
```

The access token for authenticated sessions.

Definition at line 221 of file webSocketClient.h.

5.1.4.2 m_authenticated

```
bool webSocketClient::m_authenticated  [private]
```

Indicates whether the client is authenticated.

Definition at line 219 of file webSocketClient.h.

5.1.4.3 m_authRequestCallback

```
std::function<void()> webSocketClient::m_authRequestCallback  [private]
```

Callback function for authentication requests.

Definition at line 218 of file webSocketClient.h.

5.1.4.4 m_connected

```
bool webSocketClient::m_connected  [private]
```

Indicates whether the client is connected to the server.

Definition at line 217 of file webSocketClient.h.

5.1.4.5 m_endpoint

```
client webSocketClient::m_endpoint  [private]
```

The WebSocket endpoint.

Definition at line 214 of file webSocketClient.h.

5.1.4.6 m_eventLoopThread

```
std::thread webSocketClient::m_eventLoopThread  [private]
```

The thread running the WebSocket event loop.

Definition at line 216 of file webSocketClient.h.

5.1.4.7 m_hdl

```
websocketpp::connection_hdl webSocketClient::m_hdl  [private]
```

The connection handle.

Definition at line 215 of file webSocketClient.h.

5.1.4.8 m_lastData

```
std::map<std::string, std::string> webSocketClient::m_lastData  [private]
```

Stores the last received data for each channel.

Definition at line 222 of file webSocketClient.h.

5.1.4.9 m_subscribedChannels

```
std::set<std::string> webSocketClient::m_subscribedChannels  [private]
```

Stores the names of subscribed channels.

Definition at line 223 of file webSocketClient.h.

5.1.4.10 m_waitingForResponse

```
bool webSocketClient::m_waitingForResponse  [private]
```

Indicates whether the client is waiting for a response.

Definition at line 220 of file webSocketClient.h.

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

- src/webSocketClient.h
- src/webSocketClient.cpp

Chapter 6

File Documentation

6.1 build_and_run.sh File Reference

6.2 build_and_run.sh

Go to the documentation of this file.

```
00001 #!/bin/bash
00002
00003 # Define the project directory
00004 PROJECT_DIR="/home/ashish/dericonsole"
00005 BUILD_DIR="$PROJECT_DIR/build"
00006
00007 # Function to handle errors
00008 handle_error() {
00009     echo "Error: $1" >&2
00010     exit 1
00011 }
00012
00013 # Navigate to the project directory
00014 cd "$PROJECT_DIR" || handle_error "Failed to navigate to $PROJECT_DIR"
00015
00016 # Remove the build directory if it exists
00017 if [ -d "$BUILD_DIR" ]; then
00018     echo "Removing existing build directory..."
00019     rm -rf "$BUILD_DIR" || handle_error "Failed to remove $BUILD_DIR"
00020 fi
00021
00022 # Create the build directory
00023 echo "Creating build directory..."
00024 mkdir -p "$BUILD_DIR" || handle_error "Failed to create $BUILD_DIR"
00025
00026 # Navigate to the build directory
00027 cd "$BUILD_DIR" || handle_error "Failed to navigate to $BUILD_DIR"
00028
00029 # Run CMake and build the project
00030 echo "Running CMake..."
00031 cmake -DCMAKE_BUILD_TYPE=Debug .. || handle_error "CMake failed"
00032
00033 echo "Building the project..."
00034 cmake --build . || handle_error "Build failed"
00035
00036 # Run the executable
00037 echo "Running the executable..."
00038 ./DeriConsole || handle_error "Failed to run the executable"
00039
00040 echo "Script completed successfully!"
```

6.3 dericonsole.cpp File Reference

```
#include "webSocketClient.h"
#include "deriapi.h"
```

```
#include <fmt/core.h>
#include <iostream>
#include <thread>
#include <boost/asio.hpp>
#include <boost/asio/ssl.hpp>
```

Include dependency graph for dericonsole.cpp:

Functions

- void showMenu ()

  Displays the main menu options.
- int main ()

  Main function for the WebSocket client application.

6.3.1 Function Documentation

6.3.1.1 main()

```
int main ( )
```

Main function for the WebSocket client application.

Returns

int Returns 0 on successful execution.

Definition at line 42 of file dericonsole.cpp.

6.3.1.2 showMenu()

```
void showMenu ( )
```

Displays the main menu options.

Definition at line 22 of file dericonsole.cpp.

6.4 dericonsole.cpp

Go to the documentation of this file.

```
00001
00011 #include "webSocketClient.h"
00012 #include "deriapi.h"
00013 #include <fmt/core.h>
00014 #include <iostream>
00015 #include <thread>
00016 #include <boost/asio.hpp>
00017 #include <boost/asio/ssl.hpp>
00018
00022 void showMenu() {
00023     fmt::print("\nMenu:\n");
00024     fmt::print("1. Get Account Summary\n");
00025     fmt::print("2. Place a Buy Order\n");
00026     fmt::print("3. Place a Sell Order\n");
00027     fmt::print("4. Cancel Order\n");
00028     fmt::print("5. Get Order Book\n");
00029     fmt::print("6. Modify Order\n");
00030     fmt::print("7. View Current Positions\n");
00031     fmt::print("8. Subscribe to Channel\n");
00032     fmt::print("9. Unsubscribe from Channel\n");
00033     fmt::print("10. Exit\n");
00034     fmt::print("Enter your choice: ");
00035 }
00036
00042 int main() {
00043     webSocketClient client;
00044     std::string clientId = "Cg0f13Co";
00045     std::string clientSecret = "H3Mbcyx2D1-g11g50oevQU3ej9f7cPgRj66sPY_LHOY";
00046
00047     // Set up authentication callback
00048     client.setAuthRequestCallback([&client, clientId, clientSecret]() {
00049         std::string authRequest = deriapi::authorize(clientId, clientSecret);
00050         client.send(authRequest);
00051     });
00052
00053     // Connect to the WebSocket server
00054     std::string uri = "wss://test.deribit.com/ws/api/v2";
00055     client.connect(uri);
00056
00057     // Wait for authentication to complete
00058     while (!client.isAuthenticated()) {
00059         std::this_thread::sleep_for(std::chrono::milliseconds(100));
00060     }
00061
00062     int choice;
00063     do {
00064         showMenu();
00065         std::cin » choice;
00066
00067         switch (choice) {
00068             case 1: {
00069                 std::string currency;
00070                 fmt::print("Enter Currency: ");
00071                 std::cin » currency;
00072                 std::string accountSummaryRequest = deriapi::getAccountSummary(currency);
00073                 client.send(accountSummaryRequest);
00074                 while (client.isWaitingForResponse()) {
00075                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00076                 }
00077                 break;
00078             }
00079             case 2: {
00080                 std::string instrument;
00081                 fmt::print("Enter instrument name: ");
00082                 std::cin » instrument;
00083
00084                 int amount;
00085                 fmt::print("Enter amount: ");
00086                 std::cin » amount;
00087
00088                 std::string orderType;
00089                 fmt::print("Enter order type (limit, market, stop_limit, etc.): ");
00090                 std::cin » orderType;
00091
00092                 int price = 0;
00093                 if (orderType == "limit" || orderType == "stop_limit") {
00094                     fmt::print("Enter price: ");
00095                     std::cin » price;
00096                 }
00097
00098                 std::string timeInForce;
00099                 fmt::print("Enter time-in-force (good_til_cancelled, fill_or_kill, etc.): ");
```

```
00100                 std::cin » timeInForce;
00101
00102                 std::string label;
00103                 fmt::print("Enter label: ");
00104                 std::cin » label;
00105
00106                 std::string buyRequest = deriapi::buyOrder(instrument, amount, orderType, price,
      timeInForce, label, client.getAccessToken());
00107                 client.send(buyRequest);
00108                 while (client.isWaitingForResponse()) {
00109                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00110                 }
00111                 break;
00112             }
00113             case 3: {
00114                 std::string instrument;
00115                 fmt::print("Enter instrument name: ");
00116                 std::cin » instrument;
00117
00118                 int amount;
00119                 fmt::print("Enter amount: ");
00120                 std::cin » amount;
00121
00122                 std::string orderType;
00123                 fmt::print("Enter order type (limit, market, stop_limit): ");
00124                 std::cin » orderType;
00125
00126                 double price = 0;
00127                 if (orderType == "limit" || orderType == "stop_limit") {
00128                     fmt::print("Enter price: ");
00129                     std::cin » price;
00130                 }
00131
00132                 std::string timeInForce;
00133                 fmt::print("Enter time-in-force (e.g., good_til_cancelled): ");
00134                 std::cin » timeInForce;
00135
00136                 std::string label;
00137                 fmt::print("Enter label: ");
00138                 std::cin » label;
00139
00140                 std::string sellRequest = deriapi::sellOrder(instrument, amount, orderType, price,
      timeInForce, label, client.getAccessToken());
00141                 client.send(sellRequest);
00142                 while (client.isWaitingForResponse()) {
00143                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00144                 }
00145                 break;
00146             }
00147             case 4: {
00148                 std::string orderId;
00149                 fmt::print("Enter order id: ");
00150                 std::cin » orderId;
00151                 std::string cancelRequest = deriapi::cancelOrder(orderId);
00152                 client.send(cancelRequest);
00153                 while (client.isWaitingForResponse()) {
00154                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00155                 }
00156                 break;
00157             }
00158             case 5: {
00159                 std::string instrumentName;
00160                 int depth;
00161                 fmt::print("Enter Instrument Name (e.g., BTC-PERPETUAL): ");
00162                 std::cin » instrumentName;
00163                 fmt::print("Enter depth: (if want to skip, enter 0; default is 20): ");
00164                 std::cin » depth;
00165                 if (depth == 0) depth = 20;
00166                 std::string orderBookRequest = deriapi::getOrderBook(instrumentName, depth);
00167                 client.send(orderBookRequest);
00168                 while (client.isWaitingForResponse()) {
00169                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00170                 }
00171                 break;
00172             }
00173             case 6: {
00174                 std::string orderId;
00175                 fmt::print("Enter Order ID: ");
00176                 std::cin » orderId;
00177
00178                 int amount;
00179                 fmt::print("Enter New Amount: ");
00180                 std::cin » amount;
00181
00182                 double price;
00183                 fmt::print("Enter New Price: ");
00184                 std::cin » price;
```

```
00185
00186                 std::string timeInForce;
00187                 fmt::print("Enter Time-in-Force (e.g., good_til_cancelled): ");
00188                 std::cin » timeInForce;
00189
00190                 std::string modifyRequest = deriapi::modifyOrder(orderId, amount, price, timeInForce);
00191                 client.send(modifyRequest);
00192                 while (client.isWaitingForResponse()) {
00193                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00194                 }
00195                 break;
00196             }
00197             case 7: {
00198                 std::string currency;
00199                 fmt::print("Enter Currency (e.g., BTC): ");
00200                 std::cin » currency;
00201
00202                 std::string kind;
00203                 fmt::print("Enter Instrument Type (e.g., future, option, spot): ");
00204                 std::cin » kind;
00205
00206                 std::string positionsRequest = deriapi::getPositions(currency, kind);
00207                 client.send(positionsRequest);
00208                 while (client.isWaitingForResponse()) {
00209                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
00210                 }
00211                 break;
00212             }
00213             case 8: {
00214                 std::string channel;
00215                 fmt::print("Enter channel (e.g., ticker.BTC-USD.100ms): ");
00216                 std::cin » channel;
00217                 client.subscribe(channel);
00218                 break;
00219             }
00220             case 9: {
00221                 std::string channel;
00222                 fmt::print("Enter channel to unsubscribe: ");
00223                 std::cin » channel;
00224                 client.unsubscribe(channel);
00225                 break;
00226             }
00227             case 10:
00228                 fmt::print("Exiting...\n");
00229                 break;
00230             default:
00231                 fmt::print("Invalid choice. Please try again.\n");
00232                 break;
00233         }
00234     } while (choice > 0 && choice < 10);
00235
00236     // Close the WebSocket connection
00237     client.close();
00238     return 0;
00239 }
```

6.5 src/deriapi.cpp File Reference

Implementation of the Deribit API wrapper.

```
#include "deriapi.h"
#include "utils.h"
#include <string>
#include <fmt/core.h>
#include <nlohmann/json.hpp>
```

Include dependency graph for deriapi.cpp:

6.6 deriapi.cpp

Go to the documentation of this file.

```
00001
00009 #include "deriapi.h"
```

```
00010 #include "utils.h"
00011 #include <string>
00012 #include <fmt/core.h> // Use fmt for formatted output
00013 #include <nlohmann/json.hpp>
00014
00015 using json = nlohmann::json;
00016
00017 namespace deriapi {
00018
00030     std::string authorize(const std::string& clientId, const std::string& clientSecret) {
00031         std::string timeStamp = utils::getTimeStamp();
00032         std::string nonce = utils::getNonce();
00033         std::string clientSignature = utils::getClientSignature(clientSecret, timeStamp, nonce, "");
00034
00035         // Create authorization request JSON
00036         json authRequest = {
00037             {"jsonrpc", "2.0"},
00038             {"id", 1},
00039             {"method", "public/auth"},
00040             {"params", {
00041                 {"grant_type", "client_signature"},
00042                 {"client_id", clientId},
00043                 {"timestamp", timeStamp},
00044                 {"signature", clientSignature},
00045                 {"nonce", nonce},
00046                 {"scope", "block_rfq:read_write block_trade:read_write trade:read_write
      custody:read_write account:read_write wallet:read_write mainaccount"}
00047             }}
00048         };
00049         return authRequest.dump();
00050     }
00051
00060     std::string getAccountSummary(const std::string& currency) {
00061         json accountSummaryRequest = {
00062             {"jsonrpc", "2.0"},
00063             {"id", 2},
00064             {"method", "private/get_account_summary"},
00065             {"params", {
00066                 {"currency", currency}
00067             }}
00068         };
00069         return accountSummaryRequest.dump();
00070     }
00071
00089     std::string createOrder(const std::string& method, const std::string& instrument, int amount,
      const std::string& orderType, double price, const std::string& timeInForce, const std::string& label,
      const std::string& accessToken, bool postOnly = false) {
00090         json orderRequest = {
00091             {"jsonrpc", "2.0"},
00092             {"id", 3},
00093             {"method", method},
00094             {"params", {
00095                 {"instrument_name", instrument},
00096                 {"access_token", accessToken},
00097                 {"amount", amount},
00098                 {"type", orderType},
00099                 {"label", label},
00100                 {"time_in_force", timeInForce},
00101                 {"post_only", postOnly}
00102             }}
00103         };
00104         if (orderType == "limit" || orderType == "stop_limit") {
00105             orderRequest["params"]["price"] = price;
00106         }
00107         return orderRequest.dump();
00108     }
00109
00124     std::string buyOrder(const std::string& instrument, int amount, const std::string& orderType, int
      price, const std::string& timeInForce, const std::string& label, const std::string& accessToken) {
00125         return createOrder("private/buy", instrument, amount, orderType, price, timeInForce, label,
      accessToken);
00126     }
00127
00143     std::string sellOrder(const std::string& instrument, int amount, const std::string& orderType,
      double price, const std::string& timeInForce, const std::string& label, const std::string&
      accessToken, bool postOnly) {
00144         return createOrder("private/sell", instrument, amount, orderType, price, timeInForce, label,
      accessToken, postOnly);
00145     }
00146
00155     std::string cancelOrder(const std::string& orderId) {
00156         json cancelOrder = {
00157             {"jsonrpc", "2.0"},
00158             {"id", 4},
00159             {"method", "private/cancel"},
00160             {"params", {
00161                 {"order_id", orderId}
```

```
00162             }}
00163         };
00164         return cancelOrder.dump();
00165     }
00166 
00176     std::string getOrderBook(const std::string& instrumentName, int depth) {
00177         json orderBookRequest = {
00178             {"jsonrpc", "2.0"},
00179             {"id", 5},
00180             {"method", "public/get_order_book"},
00181             {"params", {
00182                 {"instrument_name", instrumentName},
00183                 {"depth", depth}
00184             }}
00185         };
00186         return orderBookRequest.dump();
00187     }
00188 
00202     std::string modifyOrder(const std::string& orderId, int amount, double price, const std::string&
      timeInForce, bool postOnly, bool reduceOnly) {
00203         json modifyRequest = {
00204             {"jsonrpc", "2.0"},
00205             {"id", 6},
00206             {"method", "private/edit"},
00207             {"params", {
00208                 {"order_id", orderId},
00209                 {"amount", amount},
00210                 {"price", price},
00211                 {"post_only", postOnly},
00212                 {"reduce_only", reduceOnly},
00213                 {"time_in_force", timeInForce}
00214             }}
00215         };
00216         return modifyRequest.dump();
00217     }
00218 
00228     std::string getPositions(const std::string& currency, const std::string& kind) {
00229         json positionsRequest = {
00230             {"jsonrpc", "2.0"},
00231             {"id", 7},
00232             {"method", "private/get_positions"},
00233             {"params", {
00234                 {"currency", currency},
00235                 {"kind", kind}
00236             }}
00237         };
00238         return positionsRequest.dump();
00239     }
00240 
00249     std::string subscribeToChannel(const std::string& channel) {
00250         json subscribeRequest = {
00251             {"jsonrpc", "2.0"},
00252             {"id", 8},
00253             {"method", "public/subscribe"},
00254             {"params", {
00255                 {"channels", {channel}}
00256             }}
00257         };
00258         return subscribeRequest.dump();
00259     }
00260 
00269     std::string unsubscribeFromChannel(const std::string& channel) {
00270         json unsubscribeRequest = {
00271             {"jsonrpc", "2.0"},
00272             {"id", 9},
00273             {"method", "public/unsubscribe"},
00274             {"params", {
00275                 {"channels", {channel}}
00276             }}
00277         };
00278         return unsubscribeRequest.dump();
00279     }
00280 }
```

6.7 src/deriapi.h File Reference

Header file for the Deribit API wrapper.

```
#include <string>
#include <nlohmann/json.hpp>
```

Include dependency graph for deriapi.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace deriapi

Typedefs

- using json = nlohmann::json

Functions

- std::string deriapi::authorize (const std::string &clientId, const std::string &clientSecret)

  Authorizes the client using client ID and secret.
- std::string deriapi::getAccountSummary (const std::string &currency)

  Retrieves the account summary for a specific currency.
- std::string deriapi::buyOrder (const std::string &instrument, int amount, const std::string &orderType, int price, const std::string &timeInForce, const std::string &label, const std::string &accessToken)

  Creates a buy order request.
- std::string deriapi::cancelOrder (const std::string &orderId)

  Creates a cancel order request.
- std::string deriapi::getOrderBook (const std::string &instrumentName, int depth)

  Creates a request to retrieve the order book for a specific instrument.
- std::string deriapi::modifyOrder (const std::string &orderId, int amount, double price, const std::string &timeInForce, bool postOnly, bool reduceOnly)

  Creates a request to modify an existing order.
- std::string deriapi::getPositions (const std::string &currency, const std::string &kind)

  Creates a request to retrieve positions for a specific currency and kind.
- std::string deriapi::sellOrder (const std::string &instrument, int amount, const std::string &orderType, double price, const std::string &timeInForce, const std::string &label, const std::string &accessToken, bool postOnly)

  Creates a sell order request.
- std::string deriapi::subscribeToChannel (const std::string &channel)

  Creates a request to subscribe to a WebSocket channel.
- std::string deriapi::unsubscribeFromChannel (const std::string &channel)

  Creates a request to unsubscribe from a WebSocket channel.

6.7.1 Detailed Description

Header file for the Deribit API wrapper.

This file defines the functions to interact with the Deribit API, including authorization, account management, order placement, and WebSocket subscriptions.

Definition in file deriapi.h.

6.7.2 Typedef Documentation

6.7.2.1 json

```
using json = nlohmann::json
```

Definition at line 15 of file deriapi.h.

6.8 deriapi.h

Go to the documentation of this file.

```
00001
00009 #ifndef DERIAPI_H
00010 #define DERIAPI_H
00011
00012 #include <string>
00013 #include <nlohmann/json.hpp>
00014
00015 using json = nlohmann::json;
00016
00017 namespace deriapi {
00018
00030     std::string authorize(const std::string& clientId, const std::string& clientSecret);
00031
00040     std::string getAccountSummary(const std::string& currency);
00041
00056     std::string buyOrder(const std::string& instrument, int amount, const std::string& orderType, int
      price, const std::string& timeInForce, const std::string& label, const std::string& accessToken);
00057
00066     std::string cancelOrder(const std::string& orderId);
00067
00077     std::string getOrderBook(const std::string& instrumentName, int depth);
00078
00092     std::string modifyOrder(const std::string& orderId, int amount, double price, const std::string&
      timeInForce, bool postOnly = false, bool reduceOnly = false);
00093
00103     std::string getPositions(const std::string& currency, const std::string& kind = "future");
00104
00120     std::string sellOrder(const std::string& instrument, int amount, const std::string& orderType,
      double price, const std::string& timeInForce, const std::string& label, const std::string&
      accessToken, bool postOnly = false);
00121
00130     std::string subscribeToChannel(const std::string& channel);
00131
00140     std::string unsubscribeFromChannel(const std::string& channel);
00141 }
00142
00143 #endif // DERIAPI_H
```

6.9 src/utils.cpp File Reference

Implementation of utility functions for the Deribit API wrapper.

```
#include "utils.h"
#include <fmt/core.h>
#include <openssl/hmac.h>
#include <openssl/sha.h>
#include <chrono>
#include <random>
#include <iomanip>
#include <sstream>
```

Include dependency graph for utils.cpp:

Namespaces

- namespace utils

Functions

- std::string utils::getTimeStamp ()

  Generates a timestamp in milliseconds since the Unix epoch.
- std::string utils::getNonce ()

Generates a random nonce of 8 characters.

- std::string utils::toHex (const unsigned char *data, size_t length)

  Converts binary data to a hexadecimal string.

- std::string utils::hmacSha256 (const std::string &secret, const std::string &data)

  Computes the HMAC-SHA256 hash of the given data using the provided secret.

- std::string utils::getClientSignature (const std::string &clientSecret, const std::string &timeStamp, const std::string &nonce, const std::string &data)

  Generates a client signature using the provided client secret, timestamp, nonce, and data.

6.9.1 Detailed Description

Implementation of utility functions for the Deribit API wrapper.

This file contains utility functions for generating timestamps, nonces, and client signatures required for interacting with the Deribit API.

Definition in file utils.cpp.

6.10 utils.cpp

Go to the documentation of this file.

```
00001
00009 #include "utils.h"
00010 #include <fmt/core.h> // Use fmt for formatted output
00011 #include <openssl/hmac.h>
00012 #include <openssl/sha.h>
00013 #include <chrono>
00014 #include <random>
00015 #include <iomanip>
00016 #include <sstream>
00017
00018 namespace utils {
00019
00027     std::string getTimeStamp() {
00028         using namespace std::chrono;
00029         long long timeStamp =
      duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
00030         return std::to_string(timeStamp);
00031     }
00032
00040     std::string getNonce() {
00041         const std::string chars = "abcdefghijklmnopqrstuvwxyz0123456789";
00042         std::random_device rd;
00043         std::mt19937 gen(rd());
00044         std::uniform_int_distribution<char> dist(0, chars.size() - 1);
00045
00046         std::string nonce;
00047         for (int i = 0; i < 8; ++i) {
00048             nonce += chars[dist(gen)];
00049         }
00050         return nonce;
00051     }
00052
00062     std::string toHex(const unsigned char* data, size_t length) {
00063         std::ostringstream hexStream;
00064         hexStream << std::hex << std::setfill('0');
00065         for (size_t i = 0; i < length; ++i) {
00066             hexStream << std::setw(2) << (int)data[i];
00067         }
00068         return hexStream.str();
00069     }
00070
00080     std::string hmacSha256(const std::string& secret, const std::string& data) {
00081         unsigned char result[EVP_MAX_MD_SIZE];
00082         unsigned int resultLength = 0;
00083
00084         HMAC(EVP_sha256(), secret.c_str(), secret.length(),
00085              reinterpret_cast<const unsigned char*>(data.c_str()), data.length(),
00086              result, &resultLength);
```

```
00087
00088         return toHex(result, resultLength);
00089     }
00090
00103     std::string getClientSignature(const std::string& clientSecret, const std::string& timeStamp,
      const std::string& nonce, const std::string& data) {
00104         std::string stringToSign = timeStamp + "\n" + nonce + "\n" + data;
00105         return hmacSha256(clientSecret, stringToSign);
00106     }
00107 }
```

6.11 src/utils.h File Reference

Header file for utility functions used in the Deribit API wrapper.

`#include <string>`
Include dependency graph for utils.h: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace utils

Functions

- std::string utils::getTimeStamp ()

  Generates a timestamp in milliseconds since the Unix epoch.
- std::string utils::getNonce ()

  Generates a random nonce of 8 characters.
- std::string utils::getClientSignature (const std::string &clientSecret, const std::string &timeStamp, const std::string &nonce, const std::string &data)

  Generates a client signature using the provided client secret, timestamp, nonce, and data.
- std::string utils::toHex (const unsigned char *data, size_t length)

  Converts binary data to a hexadecimal string.
- std::string utils::hmacSha256 (const std::string &secret, const std::string &data)

  Computes the HMAC-SHA256 hash of the given data using the provided secret.

6.11.1 Detailed Description

Header file for utility functions used in the Deribit API wrapper.

This file defines utility functions for generating timestamps, nonces, and client signatures required for interacting with the Deribit API.

Definition in file utils.h.

6.12 utils.h

Go to the documentation of this file.

```
00001
00009 #ifndef UTILS_H
00010 #define UTILS_H
00011
00012 #include <string>
00013
00014 namespace utils {
00015
00023     std::string getTimeStamp();
00024
00032     std::string getNonce();
00033
00046     std::string getClientSignature(const std::string& clientSecret, const std::string& timeStamp,
      const std::string& nonce, const std::string& data = "");
00047
00057     std::string toHex(const unsigned char* data, size_t length);
00058
00068     std::string hmacSha256(const std::string& secret, const std::string& data);
00069 }
00070
00071 #endif // UTILS_H
```

6.13 src/webSocketClient.cpp File Reference

Implementation of the WebSocket client for interacting with a WebSocket server.

```
#include "webSocketClient.h"
#include "deriapi.h"
#include <fmt/core.h>
#include <iostream>
#include <thread>
#include <boost/asio.hpp>
#include <boost/asio/ssl.hpp>
```

Include dependency graph for webSocketClient.cpp:

6.13.1 Detailed Description

Implementation of the WebSocket client for interacting with a WebSocket server.

Definition in file webSocketClient.cpp.

6.14 webSocketClient.cpp

Go to the documentation of this file.

```
00001
00006 #include "webSocketClient.h"
00007 #include "deriapi.h"
00008 #include <fmt/core.h> // Use fmt for formatted output
00009 #include <iostream>
00010 #include <thread>
00011 #include <boost/asio.hpp>
00012 #include <boost/asio/ssl.hpp>
00013
00019 webSocketClient::webSocketClient()
00020     : m_connected(false),
00021       m_authRequestCallback(nullptr),
00022       m_authenticated(false),
00023       m_waitingForResponse(false) {
00024     // Disable logging for cleaner output
00025     m_endpoint.clear_access_channels(websocketpp::log::alevel::all);
```

```
00026     m_endpoint.clear_error_channels(websocketpp::log::elevel::all);
00027
00028     // Initialize ASIO and start perpetual loop
00029     m_endpoint.init_asio();
00030     m_endpoint.start_perpetual();
00031
00032     // Set up TLS handler
00033     m_endpoint.set_tls_init_handler([this](websocketpp::connection_hdl) {
00034         return
      websocketpp::lib::make_shared<boost::asio::ssl::context>(boost::asio::ssl::context::tlsv12);
00035     });
00036
00037     // Set up event handlers
00038     m_endpoint.set_open_handler([this](auto hdl) { this->on_open(&m_endpoint, hdl); });
00039     m_endpoint.set_fail_handler([this](auto hdl) { this->on_fail(&m_endpoint, hdl); });
00040     m_endpoint.set_close_handler([this](auto hdl) { this->on_close(&m_endpoint, hdl); });
00041     m_endpoint.set_message_handler([this](auto hdl, auto msg) { this->on_message(&m_endpoint, hdl,
      msg); });
00042 }
00043
00049 webSocketClient::~webSocketClient() {
00050     m_endpoint.stop_perpetual();
00051     if (m_connected) {
00052         close();
00053     }
00054 }
00055
00061 void webSocketClient::setAuthRequestCallback(std::function<void()> callback) {
00062     m_authRequestCallback = callback;
00063 }
00064
00070 void webSocketClient::send(const std::string& message) {
00071     websocketpp::lib::error_code ec;
00072     m_endpoint.send(m_hdl, message, websocketpp::frame::opcode::text, ec);
00073     if (ec) {
00074         fmt::print(stderr, "Send error: {}\n", ec.message());
00075     }
00076 }
00077
00083 void webSocketClient::connect(const std::string& uri) {
00084     websocketpp::lib::error_code ec;
00085     client::connection_ptr con = m_endpoint.get_connection(uri, ec);
00086     if (ec) {
00087         fmt::print(stderr, "Connection error: {}\n", ec.message());
00088         return;
00089     }
00090
00091     m_endpoint.connect(con);
00092     m_eventLoopThread = std::thread([this]() { m_endpoint.run(); });
00093 }
00094
00098 void webSocketClient::close() {
00099     if (m_connected) {
00100         websocketpp::lib::error_code ec;
00101         m_endpoint.close(m_hdl, websocketpp::close::status::normal, "Closing Connection", ec);
00102         if (ec) {
00103             fmt::print(stderr, "Close error: {}\n", ec.message());
00104             return;
00105         }
00106         m_connected = false;
00107     }
00108     m_endpoint.stop_perpetual();
00109     m_endpoint.stop();
00110
00111     if (m_eventLoopThread.joinable()) {
00112         m_eventLoopThread.join();
00113     }
00114 }
00115
00122 void webSocketClient::on_open(client* c, websocketpp::connection_hdl hdl) {
00123     fmt::print("Connection opened!\n");
00124     m_hdl = hdl;
00125     m_connected = true;
00126     if (m_authRequestCallback) {
00127         m_authRequestCallback();
00128     }
00129 }
00130
00137 void webSocketClient::on_fail(client* c, websocketpp::connection_hdl hdl) {
00138     fmt::print(stderr, "Connection failed!\n");
00139     m_connected = false;
00140 }
00141
00148 void webSocketClient::on_close(client* c, websocketpp::connection_hdl hdl) {
00149     fmt::print("Connection closed!\n");
00150     m_connected = false;
00151 }
```

```
00152
00158 void webSocketClient::subscribe(const std::string& channel) {
00159     std::string subscribeRequest = deriapi::subscribeToChannel(channel);
00160     send(subscribeRequest);
00161     m_subscribedChannels.insert(channel);
00162     m_lastData[channel] = "";
00163 }
00164
00170 void webSocketClient::unsubscribe(const std::string& channel) {
00171     fmt::print("Unsubscribed from channel: {}\n", channel);
00172     std::string unsubscribeRequest = deriapi::unsubscribeFromChannel(channel);
00173     send(unsubscribeRequest);
00174     m_subscribedChannels.erase(channel);
00175     m_lastData.erase(channel);
00176     fmt::print("Unsubscribed from channel: {}\n", channel);
00177 }
00178
00185 void webSocketClient::handleSubscriptionMessage(const std::string& channel, const nlohmann::json&
      data) {
00186     try {
00187         if (channel.find("ticker") != std::string::npos) {
00188             // Handle ticker data
00189             if (data.is_object()) {
00190                 fmt::print("Ticker Update ({}): {}\n", channel, data.dump(2));
00191             } else if (data.is_number() || data.is_string() || data.is_boolean()) {
00192                 fmt::print("Ticker Update ({}): {}\n", channel, data.dump(2));
00193             } else {
00194                 fmt::print(stderr, "Unexpected data type for ticker channel '{}'.\n", channel);
00195             }
00196         } else if (channel.find("trades") != std::string::npos) {
00197             // Handle trades data
00198             if (data.is_array()) {
00199                 fmt::print("Trade Update ({}): {}\n", channel, data.dump(2));
00200             } else {
00201                 fmt::print(stderr, "Unexpected data type for trades channel '{}'.\n", channel);
00202             }
00203         } else if (channel.find("book") != std::string::npos) {
00204             // Handle order book data
00205             if (data.is_object()) {
00206                 fmt::print("Order Book Update ({}): {}\n", channel, data.dump(2));
00207             } else {
00208                 fmt::print(stderr, "Unexpected data type for book channel '{}'.\n", channel);
00209             }
00210         } else {
00211             // Handle other channels
00212             fmt::print("Update ({}): {}\n", channel, data.dump(2));
00213         }
00214     } catch (const nlohmann::json::exception& e) {
00215         fmt::print(stderr, "JSON Parsing Error in channel '{}': {}\n", channel, e.what());
00216     }
00217 }
00218
00224 void webSocketClient::on_message_auth(nlohmann::json result) {
00225     fmt::print("Authentication successful!\n");
00226 }
00227
00233 void webSocketClient::on_message_summary(nlohmann::json result) {
00234     fmt::print("\nAccount Summary:\n");
00235     fmt::print("Email: {}\n", result.value("email", "N/A"));
00236     fmt::print("Balance: {}\n", result.value("balance", 0.0));
00237     fmt::print("Currency: {}\n", result.value("currency", "N/A"));
00238     fmt::print("Equity: {}\n", result.value("equity", 0.0));
00239     fmt::print("Initial Margin: {}\n", result.value("initial_margin", 0.0));
00240     fmt::print("Maintenance Margin: {}\n", result.value("maintenance_margin", 0.0));
00241     fmt::print("Available Funds: {}\n", result.value("available_funds", 0.0));
00242     fmt::print("Margin Balance: {}\n", result.value("margin_balance", 0.0));
00243 }
00244
00250 void webSocketClient::on_message_buy(nlohmann::json order) {
00251     fmt::print("Buy Order Placed Successfully!\n");
00252     fmt::print("Order ID: {}\n", order.value("order_id", "N/A"));
00253     fmt::print("Instrument: {}\n", order.value("instrument_name", "N/A"));
00254     fmt::print("Direction: {}\n", order.value("direction", "N/A"));
00255     fmt::print("Amount: {}\n", order.value("amount", 0.0));
00256     fmt::print("Price: {}\n", order.value("price", 0.0));
00257     fmt::print("Order Type: {}\n", order.value("order_type", "N/A"));
00258     fmt::print("Order State: {}\n", order.value("order_state", "N/A"));
00259     fmt::print("Filled Amount: {}\n", order.value("filled_amount", 0.0));
00260     fmt::print("Average Price: {}\n", order.value("average_price", 0.0));
00261     fmt::print("Creation Timestamp: {}\n", order.value("creation_timestamp", 0));
00262     fmt::print("Last Update Timestamp: {}\n", order.value("last_update_timestamp", 0));
00263 }
00264
00270 void webSocketClient::on_message_cancel(nlohmann::json result) {
00271     fmt::print("Canceled Order Successfully!\n");
00272     fmt::print("Order ID: {}\n", result.value("order_id", "N/A"));
00273     fmt::print("Time in Force: {}\n", result.value("time_in_force", "N/A"));
```

```
00274     fmt::print("Order Type: {}\n", result.value("order_type", "N/A"));
00275 }
00276
00282 void webSocketClient::on_message_orderBook(nlohmann::json result) {
00283     try {
00284         nlohmann::json orderBook = result;
00285
00286         // Print order book details
00287         fmt::print("\nOrder Book Details:\n");
00288         fmt::print("Instrument: {}\n", orderBook.value("instrument_name", "N/A")); // String
00289         fmt::print("Timestamp: {}\n", orderBook.value("timestamp", 0));           // Number
00290         fmt::print("Last Price: {}\n", orderBook.value("last_price", 0.0));       // Number
00291         fmt::print("Best Bid Price: {}\n", orderBook.value("best_bid_price", 0.0)); // Number
00292         fmt::print("Best Bid Amount: {}\n", orderBook.value("best_bid_amount", 0.0)); // Number
00293         fmt::print("Best Ask Price: {}\n", orderBook.value("best_ask_price", 0.0)); // Number
00294         fmt::print("Best Ask Amount: {}\n", orderBook.value("best_ask_amount", 0.0)); // Number
00295         fmt::print("Mark Price: {}\n", orderBook.value("mark_price", 0.0));       // Number
00296         fmt::print("Open Interest: {}\n", orderBook.value("open_interest", 0.0)); // Number
00297         fmt::print("Funding Rate (8h): {}\n", orderBook.value("funding_8h", 0.0)); // Number
00298
00299         // Handle bids
00300         fmt::print("\nBids:\n");
00301         if (orderBook.contains("bids") && orderBook["bids"].is_array()) {
00302             for (const auto& bid : orderBook["bids"]) {
00303                 if (bid.is_array() && bid.size() >= 2) {
00304                     fmt::print("Price: {}, Amount: {}\n", bid[0].get<double>(), bid[1].get<double>());
00305                 }
00306             }
00307         } else {
00308             fmt::print("No bids found.\n");
00309         }
00310
00311         // Handle asks
00312         fmt::print("\nAsks:\n");
00313         if (orderBook.contains("asks") && orderBook["asks"].is_array()) {
00314             for (const auto& ask : orderBook["asks"]) {
00315                 if (ask.is_array() && ask.size() >= 2) {
00316                     fmt::print("Price: {}, Amount: {}\n", ask[0].get<double>(), ask[1].get<double>());
00317                 }
00318             }
00319         } else {
00320             fmt::print("No asks found.\n");
00321         }
00322     } catch (const nlohmann::json::exception& e) {
00323         fmt::print(stderr, "JSON Parsing Error: {}\n", e.what());
00324     }
00325 }
00326
00332 void webSocketClient::on_message_modify(nlohmann::json result) {
00333     fmt::print("\nOrder Modified Successfully!\n");
00334     fmt::print("Order ID: {}\n", result.value("order_id", "N/A"));
00335     fmt::print("New Amount: {}\n", result.value("amount", 0.0));
00336     fmt::print("New Price: {}\n", result.value("price", 0.0));
00337     fmt::print("Order State: {}\n", result.value("order_state", "N/A"));
00338 }
00339
00345 void webSocketClient::on_message_positions(nlohmann::json result) {
00346     if (result.empty()) {
00347         fmt::print("No positions found.\n");
00348         return;
00349     }
00350     fmt::print("\nCurrent Positions:\n");
00351     for (const auto& position : result) {
00352         fmt::print("Instrument: {}\n", position.value("instrument_name", "N/A"));
00353         fmt::print("Size: {}\n", position.value("size", 0.0));
00354         fmt::print("Direction: {}\n", position.value("direction", "N/A"));
00355         fmt::print("Average Price: {}\n", position.value("average_price", 0.0));
00356         fmt::print("Mark Price: {}\n", position.value("mark_price", 0.0));
00357         fmt::print("Total Profit/Loss: {}\n", position.value("total_profit_loss", 0.0));
00358         fmt::print("Floating Profit/Loss: {}\n", position.value("floating_profit_loss", 0.0));
00359         fmt::print("Realized Profit/Loss: {}\n", position.value("realized_profit_loss", 0.0));
00360         fmt::print("Initial Margin: {}\n", position.value("initial_margin", 0.0));
00361         fmt::print("Maintenance Margin: {}\n", position.value("maintenance_margin", 0.0));
00362         fmt::print("Leverage: {}\n", position.value("leverage", 0.0));
00363         fmt::print("Estimated Liquidation Price: {}\n", position.value("estimated_liquidation_price",
      0.0));
00364         fmt::print("----------------------------\n");
00365     }
00366 }
00367
00373 void webSocketClient::on_message_sell(nlohmann::json result) {
00374     fmt::print("\nSell Order Placed Successfully!\n");
00375     fmt::print("Order ID: {}\n", result["order"].value("order_id", "N/A"));
00376     fmt::print("Instrument: {}\n", result["order"].value("instrument_name", "N/A"));
00377     fmt::print("Direction: {}\n", result["order"].value("direction", "N/A"));
00378     fmt::print("Amount: {}\n", result["order"].value("amount", 0.0));
00379     fmt::print("Price: {}\n", result["order"].value("price", 0.0));
```

```
00380     fmt::print("Order Type: {}\n", result["order"].value("order_type", "N/A"));
00381     fmt::print("Order State: {}\n", result["order"].value("order_state", "N/A"));
00382     fmt::print("Filled Amount: {}\n", result["order"].value("filled_amount", 0.0));
00383     fmt::print("Average Price: {}\n", result["order"].value("average_price", 0.0));
00384     fmt::print("Creation Timestamp: {}\n", result["order"].value("creation_timestamp", 0));
00385     fmt::print("Last Update Timestamp: {}\n", result["order"].value("last_update_timestamp", 0));
00386 }
00387
00395 void webSocketClient::on_message(client* c, websocketpp::connection_hdl hdl, client::message_ptr msg)
    {
00396     try {
00397         nlohmann::json response = nlohmann::json::parse(msg->get_payload());
00398         if (response.contains("method") && response["method"] == "subscription") {
00399             if (response.contains("params") && response["params"].is_object()) {
00400                 std::string channel;
00401                 if (response["params"].contains("channel")) {
00402                     if (response["params"]["channel"].is_string()) {
00403                         channel = response["params"]["channel"];
00404                     } else if (response["params"]["channel"].is_object() &&
    response["params"]["channel"].contains("name")) {
00405                         channel = response["params"]["channel"]["name"];
00406                     } else {
00407                         fmt::print(stderr, "Invalid channel format in JSON response.\n");
00408                         return;
00409                     }
00410                 }
00411
00412                 if (m_lastData.find(channel) == m_lastData.end()) {
00413                     fmt::print("Unsubscribed successfully from channel.\n");
00414                     return; // Channel is unsubscribed, ignore this message
00415                 }
00416
00417                 // Handle the "data" field based on its type
00418                 if (response["params"].contains("data")) {
00419                     nlohmann::json data = response["params"]["data"];
00420
00421                     // Check the type of "data"
00422                     if (data.is_object()) {
00423                         // Handle object data (e.g., book channel)
00424                         if (m_lastData[channel] != data.dump()) {
00425                             m_lastData[channel] = data.dump(); // Update the last data
00426                             handleSubscriptionMessage(channel, data); // Process the new data
00427                         }
00428                     } else if (data.is_array()) {
00429                         // Handle array data (e.g., trades channel)
00430                         if (m_lastData[channel] != data.dump()) {
00431                             m_lastData[channel] = data.dump(); // Update the last data
00432                             handleSubscriptionMessage(channel, data); // Process the new data
00433                         }
00434                     } else if (data.is_string() || data.is_number() || data.is_boolean()) {
00435                         // Handle primitive data (e.g., ticker channel)
00436                         if (m_lastData[channel] != data.dump()) {
00437                             m_lastData[channel] = data.dump(); // Update the last data
00438                             handleSubscriptionMessage(channel, data); // Process the new data
00439                         }
00440                     } else {
00441                         fmt::print(stderr, "Unexpected data type in channel '{}'.\n", channel);
00442                     }
00443                 } else {
00444                     fmt::print(stderr, "No data field found in channel '{}'.\n", channel);
00445                 }
00446             }
00447         } else if (response.contains("result")) {
00448             if (response["result"].contains("access_token")) {
00449                 m_accessToken = response["result"]["access_token"];
00450                 on_message_auth(response["result"]);
00451                 m_authenticated = true;
00452             } else if (response["result"].contains("balance")) {
00453                 on_message_summary(response["result"]);
00454             } else if (response["result"].contains("order")) {
00455                 on_message_buy(response["result"]["order"]);
00456             } else if (response["result"].contains("order_id")) {
00457                 on_message_cancel(response["result"]);
00458             } else if (response["result"].contains("bids") && response["result"].contains("asks")) {
00459                 on_message_orderBook(response["result"]);
00460             } else if (response["result"].contains("order_id")) {
00461                 on_message_modify(response["result"]);
00462             } else if (response["result"].is_array()) {
00463                 on_message_positions(response["result"]);
00464             } else if (response["result"].contains("order")) {
00465                 on_message_sell(response["result"]);
00466             }
00467         } else if (response.contains("error")) {
00468             fmt::print(stderr, "Error: {}\n", response["error"].value("message", "Unknown error"));
00469         }
00470     } catch (const nlohmann::json::exception& e) {
00471         fmt::print(stderr, "Error parsing JSON response: {}\n", e.what());
```

```
00472     }
00473 }
00474
00480 bool webSocketClient::isAuthenticated() const {
00481     return m_authenticated;
00482 }
00483
00489 bool webSocketClient::isWaitingForResponse() const {
00490     return m_waitingForResponse;
00491 }
00492
00498 std::string webSocketClient::getAccessToken() const {
00499     return m_accessToken;
00500 }
```

6.15 src/webSocketClient.h File Reference

Header file for the WebSocket client class.

```
#include <websocketpp/config/asio_client.hpp>
#include <websocketpp/client.hpp>
#include <websocketpp/common/thread.hpp>
#include <websocketpp/common/memory.hpp>
#include <nlohmann/json.hpp>
#include <fmt/core.h>
#include <iostream>
#include <thread>
#include <map>
#include <set>
#include <boost/asio.hpp>
#include <boost/asio/ssl.hpp>
```

Include dependency graph for webSocketClient.h: This graph shows which files directly or indirectly include this file:

Classes

- class webSocketClient

  A WebSocket client for interacting with a WebSocket server.

Typedefs

- typedef websocketpp::client< websocketpp::config::asio_tls_client > client
- typedef websocketpp::lib::shared_ptr< boost::asio::ssl::context > context_ptr

6.15.1 Detailed Description

Header file for the WebSocket client class.

This file defines the `webSocketClient` class, which provides functionality to connect to a WebSocket server, send and receive messages, and handle various WebSocket events such as open, close, and message reception.

Definition in file webSocketClient.h.

6.15.2 Typedef Documentation

6.15.2.1 client

```
typedef websocketpp::client<websocketpp::config::asio_tls_client> client
```

Definition at line 30 of file webSocketClient.h.

6.15.2.2 context_ptr

```
typedef websocketpp::lib::shared_ptr<boost::asio::ssl::context> context_ptr
```

Definition at line 31 of file webSocketClient.h.

6.16 webSocketClient.h

Go to the documentation of this file.

```
00001
00010 #ifndef WEBSOCKETCLIENT_H
00011 #define WEBSOCKETCLIENT_H
00012
00013 #include <websocketpp/config/asio_client.hpp>
00014 #include <websocketpp/client.hpp>
00015 #include <websocketpp/common/thread.hpp>
00016 #include <websocketpp/common/memory.hpp>
00017 #include <nlohmann/json.hpp>
00018 #include <fmt/core.h> // Use fmt for formatted output
00019 #include <iostream>
00020 #include <thread>
00021 #include <map>
00022 #include <set>
00023 #include <boost/asio.hpp>
00024 #include <boost/asio/ssl.hpp>
00025
00026 using websocketpp::lib::placeholders::_1;
00027 using websocketpp::lib::placeholders::_2;
00028 using websocketpp::lib::bind;
00029
00030 typedef websocketpp::client<websocketpp::config::asio_tls_client> client;
00031 typedef websocketpp::lib::shared_ptr<boost::asio::ssl::context> context_ptr;
00032
00041 class webSocketClient {
00042 public:
00046     webSocketClient();
00047
00053     ~webSocketClient();
00054
00060     void setAuthRequestCallback(std::function<void()> callback);
00061
00067     void send(const std::string& message);
00068
00074     void connect(const std::string& uri);
00075
00079     void close();
00080
00086     void subscribe(const std::string& channel);
00087
00093     void unsubscribe(const std::string& channel);
00094
00100     bool isAuthenticated() const;
00101
00107     bool isWaitingForResponse() const;
00108
00114     std::string getAccessToken() const;
00115
00116 private:
00123     void on_open(client* c, websocketpp::connection_hdl hdl);
00124
00131     void on_fail(client* c, websocketpp::connection_hdl hdl);
00132
00139     void on_close(client* c, websocketpp::connection_hdl hdl);
```

```
00140
00148     void on_message(client* c, websocketpp::connection_hdl hdl, client::message_ptr msg);
00149
00156     void handleSubscriptionMessage(const std::string& channel, const nlohmann::json& data);
00157
00163     void on_message_auth(nlohmann::json result);
00164
00170     void on_message_summary(nlohmann::json result);
00171
00177     void on_message_buy(nlohmann::json order);
00178
00184     void on_message_cancel(nlohmann::json result);
00185
00191     void on_message_orderBook(nlohmann::json result);
00192
00198     void on_message_modify(nlohmann::json result);
00199
00205     void on_message_positions(nlohmann::json result);
00206
00212     void on_message_sell(nlohmann::json result);
00213
00214     client m_endpoint;
00215     websocketpp::connection_hdl m_hdl;
00216     std::thread m_eventLoopThread;
00217     bool m_connected;
00218     std::function<void()> m_authRequestCallback;
00219     bool m_authenticated;
00220     bool m_waitingForResponse;
00221     std::string m_accessToken;
00222     std::map<std::string, std::string> m_lastData;
00223     std::set<std::string> m_subscribedChannels;
00224 };
00225
00226 #endif // WEBSOCKETCLIENT_H
```

Index