## cli\_wallet - main() Code flow

```
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <fc/io/json.hpp>
#include <fc/io/stdio.hpp>
#include <fc/network/http/server.hpp>
#include <fc/network/http/websocket.hpp>
#include <fc/rpc/cli.hpp>
#include <fc/rpc/http_api.hpp>
#include <fc/rpc/websocket_api.hpp>
#include <fc/smart_ref_impl.hpp>
#include <graphene/app/api.hpp>
#include <graphene/chain/config.hpp>
#include <graphene/chain/protocol/protocol.hpp>
#include <graphene/egenesis/egenesis.hpp>
#include <graphene/utilities/key_conversion.hpp>
#include <graphene/wallet/wallet.hpp>
#include <fc/interprocess/signals.hpp>
#include <boost/program_options.hpp>
#include <fc/log/console_appender.hpp>
#include <fc/log/file_appender.hpp>
#include <fc/log/logger.hpp>
#include <fc/log/logger_config.hpp>
#include <graphene/utilities/git_revision.hpp>
#include <boost/version.hpp>
#include <boost/algorithm/string/replace.hpp>
#include <websocketpp/version.hpp>
#ifdef WIN32
# include <signal.h>
# include <csignal>
#endif
```

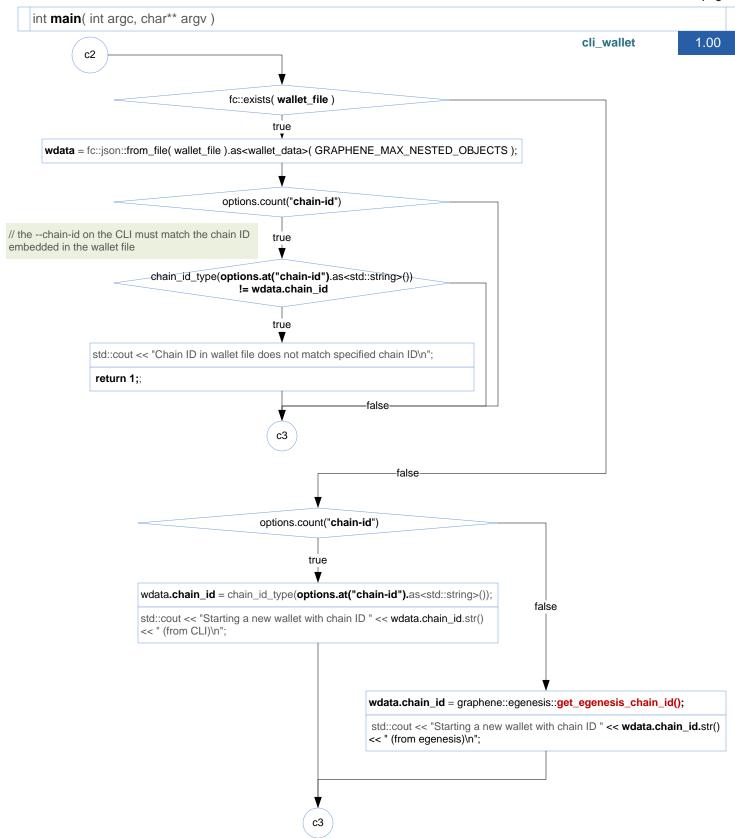
```
using namespace graphene::app;
using namespace graphene::chain;
using namespace graphene::utilities;
using namespace graphene::wallet;
using namespace std;
namespace bpo = boost::program_options;
```

```
int main(int argc, char** argv)
                                                                                                                   cli wallet
                                                                                                                                            1.00
                                 start
  try
  boost::program_options::options_description opts;
  opts.add_options()
        ("help,h", "Print this help message and exit.")
        ("server-rpc-endpoint,s", bpo::value<string>()->implicit_value("ws://127.0.0.1:8090"), "Server websocket RPC endpoint")
         "server-rpc-user,u", bpo::value<string>(), "Server Username")
        ("server-rpc-password,p", bpo::value<string>(), "Server Password")
        ("rpc-endpoint,r", bpo::value<string>()->implicit_value("127.0.0.1:8091"), "Endpoint for wallet websocket RPC to listen on")
        ("rpc-tls-endpoint,t", bpo::value<string>()->implicit_value("127.0.0.1:8092"), "Endpoint for wallet websocket TLS RPC to listen on")
        ("rpc-tls-certificate,c", bpo::value<string>()->implicit_value("server.pem"), "PEM certificate for wallet websocket TLS RPC")
        ("rpc-http-endpoint,H", bpo::value<string>()->implicit_value("127.0.0.1:8093"), "Endpoint for wallet HTTP RPC to listen on")
        ("daemon,d", "Run the wallet in daemon mode")
        ("wallet-file,w", bpo::value<string>()->implicit_value("wallet.json"), "wallet to load")
        ("chain-id", bpo::value<string>(), "chain ID to connect to")
        ("version,v", "Display version information");
  bpo::variables_map options;
  bpo::store( bpo::parse_command_line(argc, argv, opts), options );
                                                           options.count("help")
                                                                      true
                                                                                                        false
                                             std::cout << opts << "\n";
                                              return 0
                                                              options.count("version")
                                                                        true
     std::cout << "Version: " << graphene::utilities::git_revision_description << "\n";
                                                                                                                                                     false
     std::cout << "SHA: " << graphene::utilities::git_revision_sha << "\n";
     std::cout << "Timestamp: " << fc::get_approximate_relative_time_string(fc::time_point_sec(graphene::utilities::git_revision_unix_timestamp))
     << "\n";
     std::cout << "SSL: " << OPENSSL_VERSION_TEXT << "\n";
     std::cout << "Boost: " << boost::replace_all_copy(std::string(BOOST_LIB_VERSION), "_", ".") << "\n";
     std::cout << "Websocket++: " << websocketpp::major_version << "." << websocketpp::minor_version << "." << websocketpp::patch_version
     << "\n";
     return 0;
```

cli\_wallet

1.00

```
fc::path data_dir;
fc::logging_config cfg;
fc::path log_dir = data_dir / "logs";
fc::file_appender::config ac;
ac.filename
                     = log_dir / "rpc" / "rpc.log";
ac.flush
                     = true;
ac.rotate
                     = true;
ac.rotation_interval = fc::hours(1);
ac.rotation_limit
                      = fc::days( 1 );
std::cout << "Logging RPC to file: " << (data_dir / ac.filename).preferred_string() << "\n";
\textbf{cfg}. appenders. push\_back (fc::appender\_config( "\textbf{default}", "\textbf{console}", fc::variant (fc::console\_appender::config(), 20))); \\
cfg.appenders.push_back(fc::appender_config( "rpc", "file", fc::variant(ac, 5)));
cfg.loggers = { fc::logger_config("default"), fc::logger_config( "rpc") };
cfg.loggers.front().level = fc::log_level::info;
cfg.loggers.front().appenders = {"default"};
cfg.loggers.back().level = fc::log_level::debug;
cfg.loggers.back().appenders = {"rpc"};
idump( (key_to_wif( committee_private_key ) ) );
fc::ecc::private_key nathan_private_key = fc::ecc::private_key::regenerate(fc::sha256::hash(string("nathan")));
public_key_type nathan_pub_key = nathan_private_key.get_public_key();
idump( (nathan_pub_key) );
idump( (key_to_wif( nathan_private_key ) ) )
wallet_data wdata;
fc::path wallet_file( options.count("wallet-file") ? options.at("wallet-file").as<string>(): "wallet.json");
```



1.00

```
int main(int argc, char** argv)
                                                                                                               cli wallet
                                                                                                                                      1.00
     с4
                                                 wapiptr->is_new()
                                                       true
                                                                                                                false
       std::cout << "Please use the set_password method to initialize a new wallet before continuing\n";
       wallet_cli->set_prompt( "new >>> " );
                                                                                             wallet_cli->set_prompt( "locked >>> " );
  boost::signals2::scoped_connection locked_connection(wapiptr->lock_changed.connect([&](bool locked) {
       wallet_cli->set_prompt( locked ? "locked >>> " : "unlocked >>> " );
     }))
       auto _websocket_server = std::make_shared<fc::http::websocket_server>();
                                           options.count("rpc-endpoint")
                                                         teue
   _websocket_server->on_connection([&wapi]( const fc::http::websocket_connection_ptr& c ){
         auto wsc = std::make_shared<fc::rpc::websocket_api_connection>(*c, GRAPHENE_MAX_NESTED_OBJECTS);
         wsc->register_api(wapi);
         c->set_session_data( wsc );
       });
  ilog( "Listening for incoming RPC requests on ${p}", ("p", options.at("rpc-endpoint").as<string>() ));
   _websocket_server->listen( fc::ip::endpoint::from_string(options.at("rpc-endpoint").as<string>()) );
   _websocket_server->start_accept();
                                                                                   false-
  string cert_pem = "server.pem";
                                                            teue
                                             options.count( "rpc-tls-certificate" )
                          cert_pem = options.at("rpc-tls-certificate").as<string>();
       auto _websocket_tls_server = std::make_shared<fc::http::websocket_tls_server>(cert_pem);
                                         options.count("rpc-tls-endpoint")
                                                         lteue<sub>1</sub>
   _websocket_tls_server->on_connection([&wapi]( const fc::http::websocket_connection_ptr& c ){
         auto wsc = std::make_shared<fc::rpc::websocket_api_connection>(*c, GRAPHENE_MAX_NESTED_OBJECTS);
         wsc->register_api(wapi);
         c->set_session_data( wsc );
                                                                                                                                  false
       })
  llog("Listening for incoming TLS RPC requests on ${p}", ("p", options.at("rpc-tls-endpoint").as<string>() ));
   _websocket_tls_server->listen( fc::ip::endpoint::from_string(options.at("rpc-tls-endpoint").as<string>()) );
    _websocket_tls_server->start_accept();
```

```
int main(int argc, char** argv)
                                                                                                                                                 1.00
                                                                                                                         cli_wallet
                    с5
                 auto _http_server = std::make_shared<fc::http::server>();
                                                   options.count("rpc-http-endpoint")
                                                                   teue
            ilog( "Listening for incoming HTTP RPC requests on ${p}", ("p", options.at("rpc-http-endpoint").as<string>() ));
             _http_server->listen( fc::ip::endpoint::from_string( options.at( "rpc-http-endpoint" ).as<string>() ) );
// due to implementation, on_request() must come AFTER listen()
                http_server->on_request(
                      [&wapi]( const fc::http::request& req, const fc::http::server::response& resp )
                       std::shared_ptr< fc::rpc::http_api_connection > conn =
                         std::make_shared< fc::rpc::http_api_connection >( GRAPHENE_MAX_NESTED_OBJECTS );
                       conn->register_api( wapi );
                       conn->on_request( req, resp );
                      });
                                                                                     false
                                              !options.count( "daemon"
                                                         true
                                                                                                          false
                             wallet_cli->register_api( wapi );
                             wallet_cli->start();
                             wallet_cli->wait();
                                                                   fc::promise<int>::ptr exit_promise = new fc::promise<int>("UNIX Signal Handler");
                                                                   fc::set_signal_handler([&exit_promise](int signal) {
                                                                         exit_promise->set_value(signal);
                                                                       }, SIGINT);
                                                                   ilog( "Entering Daemon Mode, ^C to exit" );
                                                                   exit_promise->wait();
                          wapi->save_wallet_file(wallet_file.generic_string());
                          locked_connection.disconnect();
                          closed_connection.disconnect();
      catch (const fc::exception& e)
       std::cout << e.to_detail_string() << "\n";
                                                                   return 0;
       return -1;
                                                                                       end
```