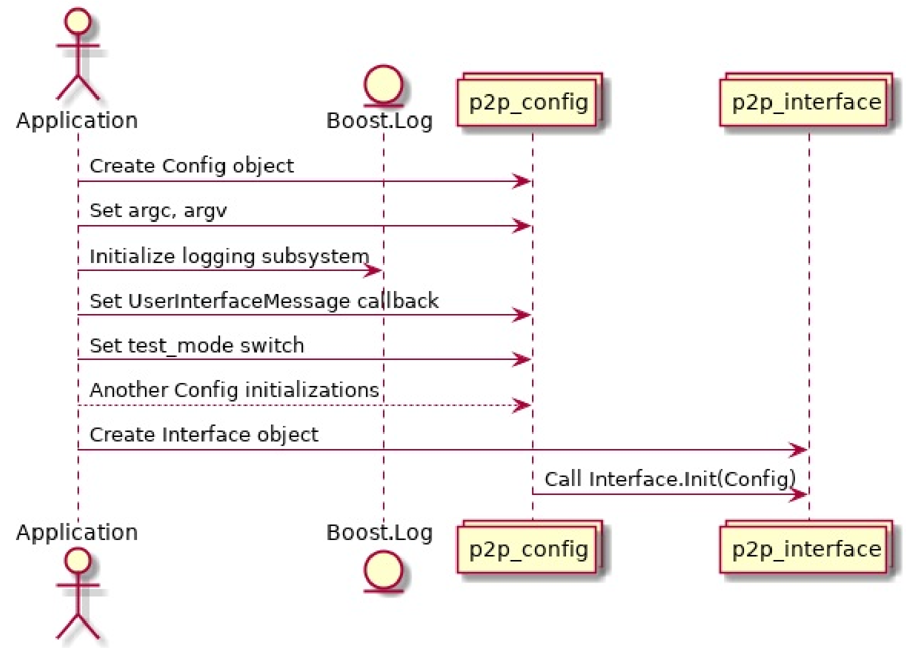
P2p: options and feedback

1. Overview

In this document, we will consider all parts of the p2p subsystem, which are responsible for passing options to the p2p module and printing debug information for the user. First of all, these are command-line options of the logos_core program which are related to the p2p subsystems. Secondly, it is a logging system that is built into the general logging system of the logos_core program. Thirdly, these are the user interface (UI) messages which can appear both during initialization and during normal program operation. Finally, we will consider the p2p-standalone test application, its command-line arguments, input and output of information from it.

2. Initialization

In this section, we describe how to initialize the p2p subsystem in the part related to this document. To initialize other parts, one need to refer to documents p2p-databases.pdf and p2p-network-stack.pdf. First, one need to create an instance of the class p2p_config and initialize its fields. In order to transfer options to the p2p module, one need to initialize the argc and argv fields, see 3.2 for more details. Then, one need to initialize the boost::log subsystem for logging purposes, see section 4.1 for details. Next, one need to set the userInterfaceMessage() callback, through which the p2p subsystem will display UI messages, see section 5 for details. One also need to set the variable test_mode to the value of true iff the p2p module is used in the unit test. In this case, the network subsystem of the p2p module will not be activated. Finally, one need to call Init function of p2p_interface and pass config to it.

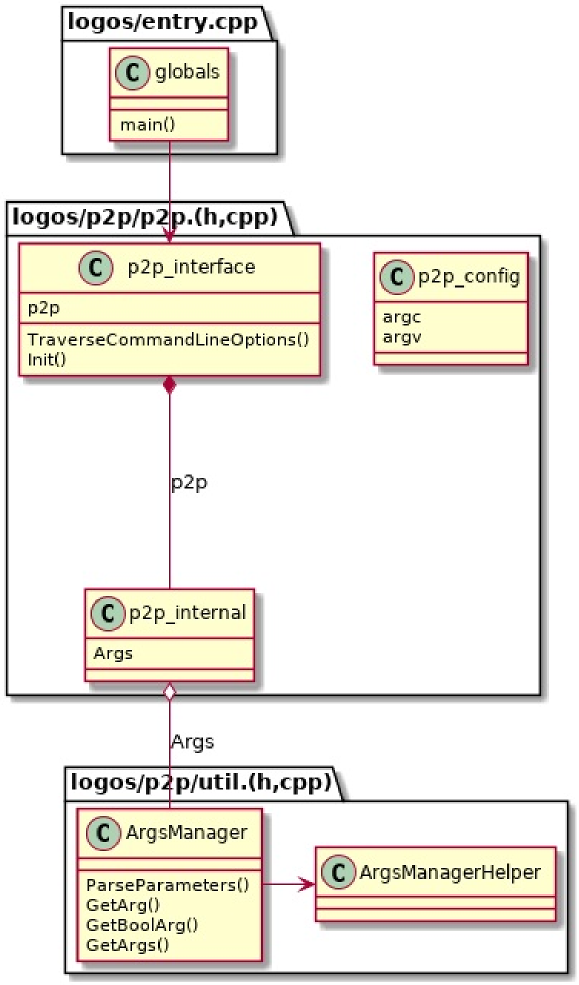


3. Command-line options

3.1. Classes

The main class responsible for handling command line options is the class ArgsManager which is described in the file logos/p2p/utils.h. For internal purposes it uses the class ArgsManagerHelper described in the file logos/p2p/utils.cpp. The classes p2p_config, p2p_interface described in the file logos/p2p/p2p.h and the auxiliary class p2p_internal described in the file logos/p2p/p2p.cpp also deals with command line options. Finally, the main() function of the logos_core project which is defined in the file logos/entry.cpp and not related to classes also works with command line options.

The fields argc, argv of the class p2p_config are used to pass options to the p2p subsystem in the same way as the options are passed to the function main(). Further, there is a static method TreverseCommandLineOptions() in the class p2p_interface which allows one to walk through all possible options of the module p2p and for each of them call a callback. Finally, the main parsing of command line options occurs in the class p2p_internal. This class, in particular, contains an instance Args of the class ArgsManager and all work with command line options is done through this variable.



3.2. Transfer of options to the p2p module

Options, both related to p2p and not, are initially passed to the main() function. The main() function firstly calls the p2p_interface::TraverseCommandLineOptions() method to get a list of all possible options of the p2p module and their descriptions. This list is passed to the standard Boost functions parsing the options, which, firstly, include it in the help, and secondly, check the correctness of the command line and parse it.

Next, in the daemon mode, the main() function re-invokes the TraverseCommandLineOptions() method to iterate through all the options. At this time the p2p options are converted into the traditional Bitcoin daemon command line representation and an array of them is formed. A pointer to this array is written in the p2p_config.argv field, and the length in the p2p_config.argc field. Further, p2p_config is passed to the method p2p_interface::Init() and thus the options come in the module p2p.

In the Bitcoin representation, it is assumed that there is one minus before the option name, and if the option has an argument, then the argument is indicated after the option name and the equal sign without spaces.

For example, the option representation in the command line of the logos_core daemon:

```
--option argument
```

The traditional Bitcoin daemon representation which is transmitted to the p2p module:

```
-option=argument
```

3.3. Analysis of command line p2p options

The initial parsing of p2p options occurs in the function p2p_interface::Init(). To do this, the function calls the ArgsManager::ParseParameters() method by using the instance p2p_internal::Args of the class ArgsManager. The fields p2p_config.argc and p2p_config.argv are passed to this method. Thus, the options are loaded into the Args instance of the class ArgsManager. Further, if some function of the p2p module is interested in whether this option is present or not, and what its value is, then it calls one of the following functions of the ArgsManager class:

```
std::string GetArg (const std::string &strArg, const std::string &strDefault);
int64_t GetArg (const std::string &strArg, int64_t nDefault);
bool GetBoolArg (const std::string &strArg, bool fDefault);
std::vector<std::string> GetArgs(const std::string &strArg);
```

The first function returns the value of the option as a string, or the string strDefault if the option is not present in the command line. The second function returns the numerical value of the option or the number nDefault if the option is absent. The third function returns the boolean value of the option or the default value fDefault. Finally, the fourth function returns the vector of the option's values if the option is present several times on the command line with different values. In all these functions, the first argument is the name of the option with a minus sign in front of it.

In addition, the ArgsManager class has other capabilities. For example, you can change the value of the option, or enter a new option with its own value. However, these features can be removed after clearing the p2p module code.

3.4. List of p2p options

The list of all p2p options is in the file logos/p2p/options.h. Each option in it is defined as

```
Arg(name, description, debug, category, arguments);
```

where
- name is the option name in quotes without preceding minuses;
- description - string for help or an expression of a form strprintf(string, parameters) where the places of parameter substitution in the string is denoted using combinations %d, %s, etc as in the C function printf;
- debug field contains true if the option is used only for debugging purposes, otherwise false;
- category - one of the categories of either connection or debugging; the options of the other categories were deleted as unnecessary;
- arguments - one of three values: 0 if the option has no arguments, P2P_OPTION_ARGUMENT if the option has agument, and P2P_OPTION_MULTI, if the option can be reused with different arguments.

A brief summary of the options is also collected in the following table the columns of which correspond to the above parameters: name, arguments (-, arg or multi) and description. Three options are marked as debug-only: --dropmessagestest, --addrmantest and --mocktime.

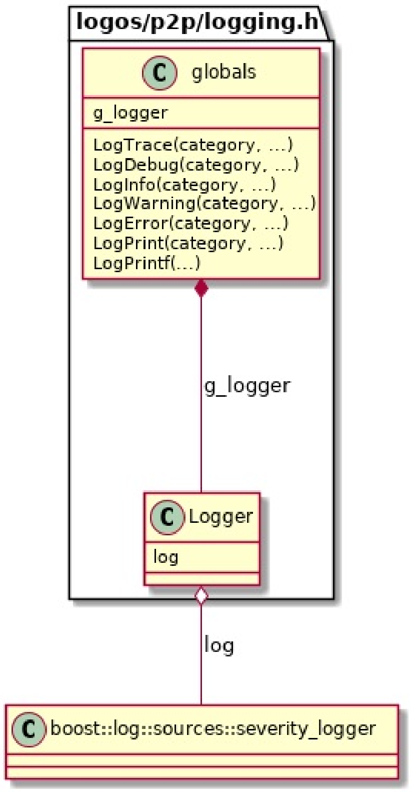
Name	**Args**	**Description**
addnode	multi	Add a node to connect to and attempt to keep the connection open
banscore	arg	Threshold for disconnecting misbehaving peers
bantime	arg	Number of seconds to keep misbehaving peers from reconnecting
bind	arg	Bind to given address and always listen on it
connect	multi	Connect only to the specified node
discover	-	Discover own IP addresses
dns	-	Allow DNS lookups for --addnode, --seednode and --connect
dnsseed	-	Query for peer addresses via DNS lookup, if low on addresses
externalip	arg	Specify your own public address
forcednsseed	-	Always query for peer addresses via DNS lookup listen
maxconnections	arg	Maintain at most arg connections to peers maxreceivebuffer
maxsendbuffer	arg	Maximum per-connection send buffer in KB
maxtimeadjustment	arg	Maximum allowed median peer time offset adjustment
maxuploadtarget	arg	Tries to keep outbound traffic under the given target in MiB per 24h
onlynet	multi	Make outgoing connections only through network ipv4 or ipv6
port	arg	Listen for connections on port arg
seednode	multi	Connect to a node to retrieve peer addresses, and disconnect
timeout	arg	Specify connection timeout in milliseconds
whitebind	arg	Bind to given address and whitelist peers connecting to it
whitelist	multi	Whitelist peers connecting from the given IP address or network
dropmessagestest	arg	Randomly drop 1 of every arg network messages
addrmantest	-	Allows to test address relay on localhost
debug	multi	Output debugging information for a category
debugexclude	multi	Exclude debugging information for a category
logips	arg	Include IP addresses in debug output, true or false
mocktime	arg	Replace actual time with arg seconds since epoch

4. Logging

4.1. Classes and principles

Logging system in the p2p module is built into the common logging system of the logos_core program or another program that uses the p2p module. For logging, the class boost::log::sources::severity_logger from the Boost library is used. The Boost logging system must be initialized before initializing the p2p module. Namely, one need to specify which file use to output the log, in what format, etc. By default logging is performed onto the standard output as it is done in the unit tests of the p2p module. An example of Boost logging initialization can be found in the program logos/p2p/test/p2p-standalone.cpp. At the same time there is no need to fill in any fields associated with logging in the p2p_config structure since the Boost logging system is global.

The main class responsible for logging in the p2p module is the class Logger defined in the file logos/p2p/logging.h. This class contains the member 'log' of the mnemonic type 'Log'. The type Log is defined in the included file ../lib/log.hpp as the severity_logger from the Boost library. Note that the file ../lib/log.hpp is the only external file from the Logos project that is included in the p2p module since it does not rely on other Logos files but includes only headers from the Boost library. The rest of the p2p module is self-sufficient. Currently, an instance *g_logger of the class Logger is a global variable and initialized before the program starts. However, after clearing the code, all global variables will propably be deleted and this variable will most likely be moved to the p2p_internal class.



4.2. Using logging system in p2p

To output to the log in the p2p module one can use the following macros defined in the file logos/p2p/logging.h:

```
LogTrace(category, ...)
LogDebug(category, ...)
```

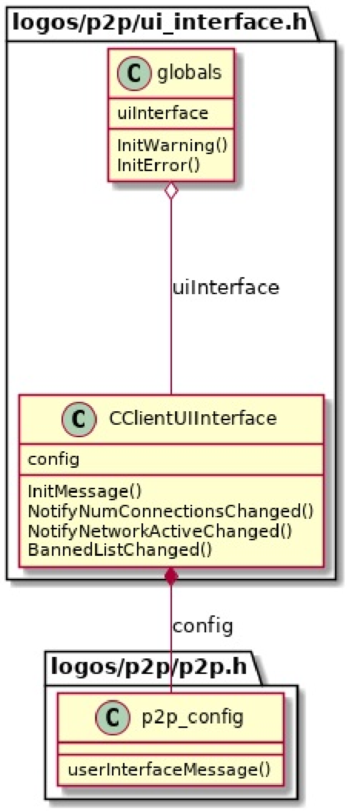
```
LogInfo(category, ...)
LogWarning(category, ...)
LogError(category, ...)
LogPrint(category, ...)
LogPrintf(...)
```

In all but the LogPrintf macro, the first argument is the category of the message, the subsequent arguments are the same as of the printf function. Currently, only three categories are used: BCLog::NET, BCLog::ADDRMAN and BCLog::RAND, the rest will propably be deleted. One can enable debugging messages for one category or another through the command line option '--debug category', for example, '--debug net'. The name of the macro contains the severity level. The minimum severity level for which the message is displayed can be set either through the Logos configuration file or during the Boost initialization process.

The macro LogPrint is the synonim of LogInfo. The last macro LogPrintf inherited from the Bitcoin project. It contains neither severity nor category. It is displayed regardless of the selected categories and has Info severity. In the process of cleaning the code, this macro will probably be deleted and all its uses will be replaced by previous macros. When outputting any message through from the p2p module, the prefix '[p2p] ' is added to it.

5. User Interface messages

5.1. Classes and principles



User interface messages are a universal mechanism that allows one to send certain important messages directly to the user, rather than putting them in a log file. Currently in the p2p module there are 4 classes of such messages:

1) initialization messages - messages about critical situations, because of which the p2p module

cannot be initialized and shuts down, or continues to work, but in a limited mode;
2) messages on changes in the number of connections - are informational;
3) network status change messages - warn the user that there is no connection with other nodes, or the connection has appeared;
4) messages about changing the list of blocked nodes - warn the user, as it may require his intervention if the ban is set incorrectly.

Before initializing the p2p module in the structure p2p_config, one need to set the userInterfaceMessage callback which will be called for each user interface message. The main class responsible for user interface messages is the class CClientUIInterface defined in the file logos/p2p/ui_interface.h. An instance of this class is the global variable uiInterface; that may be changed in the future. When the p2p module is initialized, the function p2p_interface::Init() places a pointer to the p2p_config configuration into the field uiInterface.config and thus the CClientUIInterface class can directly call the callback to send custom messages to it.

5.2. Using the UI messages

The callback p2p_config.userInterfaceMessage is a function with the following prototype:

```
void ui_callback(int type, const char *message);
```

where type is an bitmask which consists of severity of the message (one of the P2P_UI_INFO, P2P_UI_WARNING, P2P_UI_ERROR) values and, possibly, an indication that the message is an initialization message (P2P_UI_INIT). Callback can display the message in one way or another depending on its type. For example, the callback for the program logos_core ignores messages if the severity is informational, and the callback for application p2p-standalone displays all messages to the terminal.

To send a UI message from the p2p module, one can use one of the following functions, some of which are methods of the class CClientUIInterface, and some are global functions. In the future this can be unified.

```
void CClientUIInterface::InitMessage(const std::string& message);
void CClientUIInterface::NotifyNumConnectionsChanged(int newNumConnections);
void CClientUIInterface::NotifyNetworkActiveChanged(bool networkActive);
void CClientUIInterface::BannedListChanged();
void InitWarning(const std::string& str);
bool InitError(const std::string& str);
```

The first 4 methods lead to sending messages of types 1) -4). described in section 5.1. In this case the message is of informational severity. For the second message, the parameter is the new number of connections, for the third, the status of the network (true - active, false - not). The global functions InitWarning() and InitError() are intended to send an initialization message with the appropriate severity.

6. Test application

6.1. Application Overview

The p2p-standalone test application is a p2p subsystem linked together with a minimal main() function which fills in the p2p_config structure and initializes the p2p module. The application can be connected to the existing Logos network or to other similar applications in order to debug the p2p network. The application performs all the functions that the p2p module performs, namely, it establishes connections with other p2p modules, distributes messages received from other nodes to the network, maintains the network nodes database and exchanges node addresses with other nodes, saves this database to disk and reads it from disk on next run. In addition, the application displays all incoming p2p messages and distributes text messages entered from the keyboard over the p2p network.

6.2. Application launch

The p2p-standalone test application is located in the file logos/p2p/test/p2p-standalone.cpp. It is assembled together with other programs from the project logos_core when the make command is executed. This application is interactive, it needs to be launched in the terminal and after launching it receives commands from the console and displays messages on the screen. Currently, before launching the application, one need to manually create a '.logos' subdirectory in the current working directory. In the future, this deficiency is likely to be eliminated.

The p2p-standalone application accepts the same command line options as the p2p module, see section 3. However, the command line options must be set in the Bitcoin options format as described in section 3.2. Here is an example of launching an application in a frequently used situation:

```
./p2p-standalone -port=1234 -debug=net -addnode=localhost:1235 -addnode=172.1.1.163:14495
```

In this case, the application will listen for incoming connections on port 1234, will log messages related to the network, will connect to another p2p-standalone application listening to connections on the local machine on port 1235, and will connect to one of the nodes from Logos network which runs on a machine with ip address 172.1.1.163 and listens to p2p connections on the standard port 14495.

6.3. Application operation

After launch, the p2p-standalone application becomes a full member of the Logos p2p network and all p2p messages circulating in the network will come to it. The application displays all incoming messages, namely, the length of the message and its several hundred first bytes in hexadecimal format are displayed. One can also enter text messages from the console, but since they are not in the Logos message format, other Logos nodes will reject them, but other p2p-standalone applications in the same p2p network can display such text messages.

In addition, the p2p-standalone application also displays all User Interface messages outgoing from the p2p module. Finally, in addition to test messages, the following commands can be entered from the keyboard in the interactive mode:

```
exit
peers
ban <ip>
banned <ip>
```

The 'exit' command terminates the test application. The 'peers' command displays a list of all known hosts on the p2p network. The 'ban' command whose argument is the ip address of the host adds this host to the database of banned hosts. Finally, the 'banned' command whose argument is also an ip address returns a true or false depending on whether the given host is in the database of banned hosts.