Webserv

Generated by Doxygen 1.12.0

Chapter 1

webserv

Balbalsba

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 documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Client Class Reference

Public Member Functions

- **Client** (Server server, int socket)
- void **appendRequest** (char const *buffer, size_t size)
- bool **isTimeout** () const
- bool **clientIsReadyToReceive** () const
- void **handleClientRequest** ()
- const std::string & **getRequest** () const
- const Server & **getServer** () const
- void **changeServer** (Server server)

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Client.hpp
- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/src/Client.cpp

4.2 location_s Struct Reference

Public Attributes

- std::string **root**
- std::vector< std::string > **methods**
- std::string **redirect**
- bool **autoindex**
- std::string **tryFile**
- bool **hasCGI**
- std::string **cgiPath**
- std::string **cgiExtension**
- std::string **uploadTo**

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Server.hpp

4.3 Parser Class Reference

Public Member Functions

- **Parser** (int argc, char **argv)
- std::vector< Server > & **getServersVector** ()
- std::set< std::string > **getSupportedExtensions** ()

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Parser.hpp
- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/src/Parser.cpp

4.4 Server Class Reference

Public Member Functions

- **Server** (std::vector< Server > &_serversVector, std::map< std::string, std::string > tempServerConfigMap, std::vector< std::map< std::string, std::string > > tempLocationMapVector)
- void **createSocket** ()
- bool **getIsDefault** ()
- const std::string & **getHost** () const
- const std::string & **getPort** () const
- int **getSocket** () const
- const std::string & **getServerName** () const
- const std::string & **getRoot** () const
- const std::string & **getIndex** () const
- const std::string & **getErrorPage** () const
- const std::string & **getErrorResponse** () const
- size_t **getClientMaxBodySize** () const
- void **printServers** ()
- void **printLocation** (location_t loc)

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Server.hpp
- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/src/Server.cpp

4.5 Service Class Reference

Public Member Functions

- **Service** (int argc, char **argv)
- void **setup** ()
- void **launch** ()
- void **printServersInfo** ()
- void **printServiceInfo** ()

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Service.hpp
- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/src/Service.cpp

4.6 serviceInfo Struct Reference

Public Attributes

- addrinfo **parameters**
- addrinfo * **address**
- std::string **host**
- std::string **port**
- int **listeningSocketFd**
- int **clientID**
- int **serverID**
- int **connectionSocketFd**
- int **pollID**
- short **mode**
- bool **launch**

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Service.hpp

4.7 Token Struct Reference

Public Attributes

- eToken **type**
- std::string **value**

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

- /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/include/Parser.hpp

Chapter 5

File Documentation

5.1 Client.hpp

```
00001 #ifndef CLIENT_HPP
00002 #define CLIENT_HPP
00003
00004 #include "defines.hpp"
00005 #include "Server.hpp"
00006 #include "utils.hpp"
00007
00008 class Client
00009 {
00010     private:
00011         Server        _server;
00012         int         _socket;
00013         bool        _sentRequest;
00014         std::string _request;
00015         time_t      _lastRequest;
00016         std::string _resourcePath; //path to the ressource ../site/page
00017         std::string _method; //GET POST DELETE
00018
00019         std::string                         _requestPayload;
00020         std::map<std::string, std::string>  _headers;
00021
00022
00023         //befor sending request to the Client
00024
00025         void        _checkRequest();
00026         void        _checkFirstLine(std::stringstream &ss);
00027         void        _checkAndGetHeaders(std::stringstream &ss);
00028         void        _checkAndGetPayload(std::stringstream &ss);
00029         void        _checkLocation(std::string &root, std::string &resource, size_t loopCount);
00030
00031
00032         Client();
00033
00034     public:
00035         Client(Server server, int socket);
00036         ~Client();
00037
00038         void        appendRequest(char const *buffer, size_t size);
00039         bool        isTimeout() const;
00040         bool        clientIsReadyToReceive() const;
00041
00042
00043         // void     sendResponseToClient();
00044         void        handleClientRequest(); //Janneta's function
00045
00046
00047
00048
00049
00050         const std::string&  getRequest() const;
00051         const Server&         getServer() const;
00052         void                changeServer(Server server);
00053 };
00054
00055 #endif
```

5.2 defines.hpp

```
00001 #ifndef DEFINES_HPP
00002 #define DEFINES_HPP
00003
00004 // standart libraries
00005 #include <iostream>
00006 #include <csignal>
00007 #include <string>
00008 #include <cstring>
00009 #include <iomanip>
00010 #include <ctime>
00011 #include <exception>
00012 #include <map>
00013 #include <fstream>
00014 #include <sstream>
00015 #include <vector>
00016 #include <regex>
00017 #include <fcntl.h>
00018 #include <stdexcept>
00019 #include <cctype>
00020 #include <unistd.h>
00021 #include <set>
00022 #include <sys/stat.h>
00023 #include <sys/types.h>
00024 #include <dirent.h>
00025 #include <sys/socket.h>
00026 #include <netdb.h>
00027 #include <arpa/inet.h>
00028 #include <poll.h>
00029
00030 // Global variables
00031 extern bool g_shutdown;
00032
00033 //Defaut settings
00034
00035 #define POLL_TIME_OUT       200     // 200ms
00036 #define MAX_PENDING         10      // Maximum number of pending connections
00037 #define BUFFER_SIZE         2048    // 2KB
00038 #define SENT_TIMEOUT        60      // 60s
00039
00040 // Charset
00041 #define REQUEST_END         "\r\n\r\n"
00042 #define CURSOR_NEWLINE      "\r\n"
00043 #define REQUEST_HOST        "Host: "
00044
00045
00046
00047 // Custom Outputs
00048 #define RED     "\033[0;31m"
00049 #define GREEN   "\033[0;32m"
00050 #define BLUE    "\033[0;34m"
00051 #define RESET   "\033[0m"
00052 #define CLEAR   "\033c"
00053
00054
00055 // printInfo messages
00056 #define START_MSG                   "Webserv is starting..."
00057 #define END_MSG                     "Webserv shutdowned"
00058 #define SHUTDOWN_MSG                "Webserv is shutting down..."
00059 #define SET_SERVER_MSG(host, port)  "Server " + host + ":" + port + " setup complete"
00060 #define LAUNCH_MSG                  "Launching servers..."
00061 #define EMPTY_MSG                   ""
00062 #define POLLERR_MSG                 "Connection closed. Error: POLLERR"
00063 #define POLLHUP_MSG                 "Connection closed. Error: POLLHUP"
00064 #define POLLNVAL_MSG                "Connection closed. Error: POLLNVAL"
00065 #define CLOSE_MSG                   "Connection closed"
00066 #define TIMEOUT_MSG                 "Connection closed. Timeout"
00067
00068 // Parser check input errors
00069 #define ERR_ARG                         "Invalid arguments\n\tUsage: ./webserv [config_file]"
00070 #define ERR_FILE_CONF(confFile)         "'"+ confFile + "' is a invalid file\n\tFile must have a name
      and must be .conf"
00071 #define ERR_OPEN                        "Couldn't open file "
00072 #define ERR_NO_SERVER_CONFIG            "There is no bloc server in the configuration file "
00073
00074 //Parser token error
00075 #define ERR_INVALID_KEY(line)           "Invalid keyword on line: '" + line + "'"
00076 #define ERR_SEMICOLON(line)                     "Missing ';' at line '" + line +"'"
00077 #define ERR_CLOSING_BRACKETS                    "Missing closing bracket"
00078 #define ERR_OPENING_BRACKET                     "Missing opening bracket"
00079 #define ERR_SERV_FORBIDDEN_DIRECTIVE(directive)     "Directive '" + directive + "' is not allowed in
      Server block"
00080 #define ERR_LOC_FORBIDDEN_DIRECTIVE(directive)      "Directive '" + directive + "' is not allowed in
      Location block"
00081
00082 //Bloc Server error
```

```
00083 #define ERR_PORT_INPUT(port)                "'"+ port + "' is not a valid port number. Port must be a
      number between 0 and 65535"
00084 #define ERR_HOST_INPUT(host)                "'"+ host + "' is not a valid host name or ip adresse."
00085 #define ERR_DIRECTORY(path)                 "'" + path + "' is not a valid directory"
00086 #define ERR_NOT_FILE_NOT_DIR(path)          "'" + path + "' is neither a regular file nor a
      directory."
00087 #define ERR_FILE_INFO(path)                 "Error retrieving file info for: '" + path + "'"
00088 #define ERR_OPEN_DIR(path)                  "Unable to open directory: '" + path + "'"
00089 #define ERR_PATH(path)                      "'" + path + "' is not a valid path"
00090 #define ERR_WRITE_PERM(path)                "Write access denied for path: '" + path + "'"
00091 #define ERR_SERV_DIRECTIVE_MISSING(directive)   "Missing Directive '" + directive + "' in Server
      block"
00092 #define ERR_LOC_DIRECTIVE_MISSING(directive)    "Missing Directive '" + directive + "' in Location
      block"
00093 #define ERR_FILE(file)                      "Couldn't open file '" + file + "' ."
00094 #define ERR_MAX_SIZE_INPUT(size)            "'" + size + "' is not a valid size. Size must be a number
      positive or a number followed by a sufix (b - B, k - K, m - M, g - G)"
00095 #define ERR_MAX_SIZE_RANGE(size)            "'" + size + "' is not a valid size. The max value allowed
      is 10G (10737418240 bytes)"
00096 #define ERR_MAX_SIZE_CONVERSION(size)           "'" + size + "' is not a valid number."
00097 #define ERR_MAX_SIZE_CONVERSION_LONG(size)          "'" + size + "' is a number too large for long."
00098 #define ERR_INVALID_SERVER_NAME(server)     "'"+ server + "' is not a valid server name, only
      alphanumeric, hyphens, and periods are allowed "
00099
00100 //Bloc Location Error
00101 #define ERR_LOCATION(path)                      "Location's path needs to begin with a '/' "
00102 #define ERR_INVALID_METHOD(method, directive)   "The method '" + method + "' in the directive'value
      :'" + directive + "' ."
00103 #define ERR_AUTOINDEX   "The autoinde value should be 'on' or 'off'"
00104 #define ERR_ERR_CGI_DOT(extension)          "Extension '" + extension + "' must start with a point
      (.)"
00105 #define ERR_ERR_CGI_EXT(extension)          "Unsupported CGI extension: '" + extension + "'"
00106
00107
00108 //Setup Error
00109 #define ERR_SOCKET(server)              "failed to create network socket for server " + server
00110
00111
00112 // Service setServersAddress errors
00113 #define ERR_SET_SOCKET                  "setsockopt() failed: "
00114 #define ERR_GET_ADDR_INFO               "getaddrinfo() failed: "
00115 #define ERR_BIND_SOCKET                 "bind() failed: "
00116 #define ERR_LISTEN_SOCKET               "listen() failed: "
00117
00118 // Default settings
00119 #define DEFAULT_CONF        "confFiles/default.conf"
00120
00121 // Service launch errors
00122 #define ERR_POLL_FAIL                   "poll() failed"
00123 #define ERR_ACCEPT_SOCKET               "accept() failed"
00124
00125 // Server parameters
00126 #define SERVER      "server"
00127 #define LISTEN      "listen"
00128 #define HOST        "host"
00129 // #define ROOT     "root"
00130 #define INDEX       "index"
00131 #define MAX_SIZE    "client_max_body_size"
00132 #define SERVER_N    "server_name"
00133 #define ERROR_P     "error_page"
00134
00135 // Servers'Location parameters
00136 #define LOCATION    "location"
00137 #define ALLOW_M     "allow_methods"
00138 #define TRY         "try_file"
00139 #define RETURN      "return"
00140 #define AUTOID      "autoindex"
00141 #define ROOT_LOC    "root"
00142 #define UPLOAD      "upload_to"
00143 #define CGI_P       "cgi_path"
00144 #define CGI_E       "cgi_ext"
00145
00146 #endif
```

5.3 Parser.hpp

```
00001 #ifndef PARSER_HPP
00002 #define PARSER_HPP
00003
00004 #include "defines.hpp"
00005 #include "Server.hpp"
00006 #include "utils.hpp"
00007
```

```
00008 enum eToken {
00009     TK_SERVER,
00010     TK_LOCATION,
00011     TK_CLOSE_BRACKET,
00012     TK_TOKEN,
00013     TK_ERROR,
00014     TK_EMPTY,
00015     TK_COMMENT
00016 };
00017
00018 class Server;
00019 // struct location_t;
00020
00021 struct Token
00022 {
00023     eToken type;
00024     std::string value;
00025 };
00026
00027 typedef std::vector<Server> serverVector;
00028
00029 class Parser{
00030
00031     private:
00032             void    _checkInputArg(int argc, char **argv);
00033             void    _parseFile();
00034             void    _checkBracket();
00035             void    _getConfigAndInitServers();
00036             void    _getConfigFromTokens();
00037             void    _getParamFromToken(int enumToken);
00038             void    _getServerParam();
00039             void    _checkServerParam();
00040             void    _getLocationParam();
00041             void    _checkLocationParam();
00042             void    _checkDirectiveName();
00043             void    _checkDirectiveValue();
00044             void    _checkLocDirName();
00045             void    _checkLocDirValue();
00046             void    _checkConfigs();
00047             void    _delEndSemiColon(std::string& s);
00048             void    _checkPath(std::string& dirValue, bool isDir, bool hasWPer = false);
00049
00050             //Directives checking functions:
00051             void    _checkListen(std::string& dirValue);
00052
00053             void    _checkHost(std::string& dirValue);
00054             bool    _isValidIpAddress(const std::string& str);
00055             bool    _isValidHostName(const std::string& str);
00056             void    _checkRoot(std::string& dirValue);
00057             void    _checkIndex(std::string& dirValue);
00058             void    _checkMaxSize(std::string& dirValue);
00059             void    _checkServerN(std::string& dirValue);
00060             void    _checkErrorP(std::string& dirValue);
00061
00062             //Directives' Location checking functions:
00063             void    _checkLocation(std::string& dirValue);
00064             void    _checkAllowM(std::string& dirValue);
00065             void    _checkTry(std::string& dirValue);
00066             void    _checkReturn(std::string& dirValue);
00067             void    _checkAutoID(std::string& dirValue);
00068             void    _checkRootLoc(std::string& dirValue);
00069             void    _checkUpload(std::string& dirValue);
00070             void    _checkCgiP(std::string& dirValue);
00071             void    _checkCgiE(std::string& dirValue);
00072
00073
00074
00075             size_t      _nServer;
00076             size_t      _nbLine;
00077
00078
00079             std::string                                         _confFilePath;
00080             std::ifstream                                       _confFile;
00081             std::vector<Token>                                      _tokensVector;
00082
00083             std::vector<Server>                                 _serversVector;
00084             std::map<std::string, std::string>                  _tempServerConfigMap;
00085
00086             std::vector<std::map<std::string, std::string> >    _tempLocationMapVector;
00087             std::map<std::string, std::string>                  _tempLocationConfigMap;
00088             std::vector<location_t>                             _tempLocationVector;
00089             location_t                                            _tempLocation;
00090             std::string                                         _tempRootDirPath;
00091
00092
00093
00094     public:
```

```
00095             Parser(int argc, char **argv);
00096             ~Parser();
00097             std::vector<Server>& getServersVector();
00098             std::set<std::string> getSupportedExtensions();
00099
00100
00101 };
00102 #endif
```

5.4 Server.hpp

```
00001 #ifndef SERVER_HPP
00002 #define SERVER_HPP
00003
00004 #include "defines.hpp"
00005 #include "utils.hpp"
00006
00007 typedef struct location_s
00008 {
00009     std::string                 root;
00010     std::vector<std::string>    methods;
00011     std::string                 redirect;
00012     bool                        autoindex;
00013     std::string                 tryFile;
00014     bool                        hasCGI;
00015     std::string                 cgiPath;
00016     std::string                 cgiExtension;
00017     std::string                 uploadTo;
00018 } location_t;
00019
00020 class Server{
00021     private:
00022             std::string                     _serverName;
00023             std::string                     _port;
00024             std::string                     _host;
00025             std::string                     _root;
00026             std::string                     _index;
00027             long                            _clientMaxBodySize;
00028             std::string                     _errorPage;
00029             std::string                     _errorResponse;
00030             std::vector<location_t>         _tempLocationVector;
00031             bool                            _isDefault;
00032             int                             _socket;
00033
00034             bool                            _checkDefaultServer(std::vector<Server>& _serversVector);
00035             long                            _getConvertedMaxSize(std::string& maxSizeStr);
00036
00037
00038
00039             //Servers
00040             // std::vector<Server>                  _serversVector;
00041             std::map<std::string, std::string>                  _ServerConfigMap;
00042
00043
00044             //Location
00045             std::vector<std::map<std::string, std::string> >        _LocationMapVector;
00046             std::map<std::string, std::string>                  _LocationConfigMap;
00047             std::map<std::string, location_t>                   _LocationMap;
00048             void                                                _getLocationStruct();
00049
00050     public:
00051             Server(std::vector<Server>&   _serversVector, std::map<std::string, std::string>
    tempServerConfigMap, std::vector<std::map<std::string, std::string> > tempLocationMapVector);
00052             ~Server();
00053             //called from Service
00054             void    createSocket();
00055
00056             //getters
00057             bool                getIsDefault();
00058             const std::string&  getHost() const;
00059             const std::string&  getPort() const;
00060             int                 getSocket() const;
00061             const std::string&  getServerName() const;
00062             const std::string&  getRoot() const;
00063             const std::string&  getIndex() const;
00064             const std::string&  getErrorPage() const;
00065             const std::string&  getErrorResponse() const;
00066             size_t              getClientMaxBodySize() const;
00067
00068
00069
00070             //utils
00071             void                printServers();
```

```
00072             void                printLocation(location_t loc);
00073
00074
00075
00077
00078 };
00079
00080 #endif
```

5.5 Service.hpp

```
00001 #ifndef SERVICE_HPP
00002 #define SERVICE_HPP
00003
00004 #include "defines.hpp"
00005 #include "Parser.hpp"
00006 #include "Client.hpp"
00007 // #include "Service.hpp"
00008 // #include "utils.hpp"
00009
00010
00011 struct serviceInfo
00012 {
00013     addrinfo    parameters;
00014     addrinfo    *address;
00015     std::string host;
00016     std::string port;
00017     int         listeningSocketFd; //to listen - for server
00018     int         clientID;
00019     int         serverID;
00020     int         connectionSocketFd; //to connect - for client
00021     int         pollID;
00022     short       mode;
00023     bool        launch;
00024 };
00025
00026
00027
00028 /*
00029 For information:
00030
00031 struct addrinfo {
00032     int              ai_flags;      // options
00033     int              ai_family;     // adress family (ipv4, ipv6) (AF_INET, AF_INET6, etc.)
00034     int              ai_socktype;   // kind of socket (SOCK_STREAM, SOCK_DGRAM, etc.)
00035     int              ai_protocol;   // protocol (IPPROTO_TCP, IPPROTO_UDP, etc.)
00036     size_t           ai_addrlen;    // adress size
00037     struct sockaddr *ai_addr;       // socket addr
00038     char            *ai_canonname;  // canonical nam
00039     struct addrinfo *ai_next;       // pointer to the nex struct
00040 };
00041 */
00042
00043 class Service
00044 {
00045     private:
00046             std::vector<Server> _serversVector;
00047             std::vector<Client> _clientVector;
00048             std::vector<pollfd> _pollingFdVector;
00049             size_t          _defaultServers;
00050             serviceInfo      _tmpServiceInfo;
00051
00052             void            _initServiceInfo();
00053             void            _getSetupInfo(std::vector<Server>::iterator server);
00054             void            _setReuseableAddress();
00055             void            _resetTmpServiceInfo();
00056             void            _convertHostToAddress();
00057             void            _bindAddressToSocket();
00058             void            _setSocketListening();
00059             void            _addSocketToPollSockVec();
00060             void            _launch();
00061             void            _initPollingVector();
00062             void            _pollingManager();
00063             void            _getLaunchInfo(int const i);
00064             int             _getServerIndex();
00065             bool            _hasDataToRead();
00066             bool            _isServerSocket();
00067             void            _acceptConnection();
00068             void            _readDataFromClient();
00069             void            _closeConnection(std::string const &msg);
00070             bool            _hasBadRequest();
00071             void            _sendDataToClient();
00072             void            _checkRequestedServer();
```

```
00073
00074
00075     public:
00076             Service(int argc, char **argv);
00077             ~Service();
00078
00079             void    setup();
00080             void    launch();
00081             void    printServersInfo();
00082             void    printServiceInfo();
00083
00084 };
00085
00086 #endif
```

5.6 utils.hpp

```
00001 #ifndef UTILS_HPP
00002 #define UTILS_HPP
00003
00004 #include "defines.hpp"
00005
00006 /* === String === */
00007
00008 /* === Time === */
00009 std::string     getTime();
00010
00011 /* === Logs === */
00012
00013 /* === Signal === */
00014 void    signalHandler(int signum);
00015
00016 /* === PrintInfo === */
00017
00018 void printMap(const std::map<std::string, std::string>& mapToPrint);
00019 void printInfo(std::string const &s, std::string const &color);
00020
00021
00022
00023 #endif
```

5.7 webserv.hpp

```
00001 /* ************************************************************************** */
00002 /*                                                                            */
00003 /*                                                        :::      ::::::::   */
00004 /*   webserv.hpp                                        :+:      :+:    :+:   */
00005 /*                                                    +:+ +:+         +:+     */
00006 /*   By: thibault <thibault@student.42.fr>          +#+  +:+       +#+        */
00007 /*                                                +#+#+#+#+#+   +#+           */
00008 /*   Created: 2024/09/30 14:24:20 by thibault          #+#    #+#             */
00009 /*   Updated: 2024/10/26 15:48:48 by thibault         ###   ########.fr       */
00010 /*                                                                            */
00011 /* ************************************************************************** */
00012
00013 // header to include all class header
00014
00015 #ifndef WEBSERV_HPP
00016 #define WEBSERV_HPP
00017
00018 #include "Service.hpp"
00019 #include "Parser.hpp"
00020 #include "Server.hpp"
00021
00022
00023 #endif
```

5.8 /Users/thibault/kdrive/1-PROJECTS/P-42/webserv/src/Client.cpp File Reference

```
#include "Client.hpp"
```

5.8.1 Detailed Description

Author

tsanglar

Version

0.1

Date

2024-10-30

Copyright

Index