webserv

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 log Namespace Reference

Functions

- std::string logFname (void)
- std::string strTime (void)
- void timestamp (void)
- void print (const str_t &)
- void printVec (vec_str_t &, const str_t)

Variables

- const std::time_t begin = std::time(NULL)
- File history

5.1.1 Function Documentation

5.1.1.1 logFname()

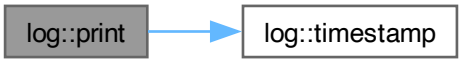
```
std::string log::logFname (
            void )
```

Here is the call graph for this function:

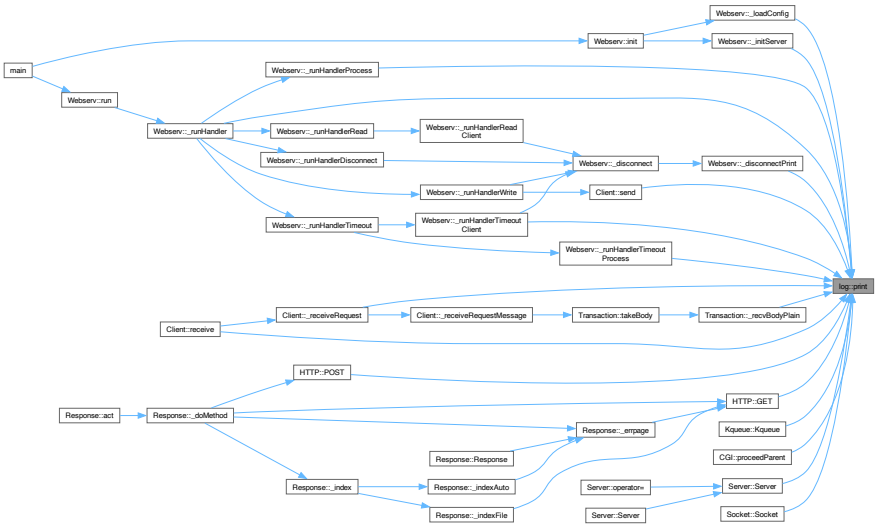
5.1.1.2 print()

```
void log::print (
            const str_t & msg)
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.1.3 printVec()

```
void log::printVec (
            vec_str_t & ,
            const str_t )
```

5.1.1.4 strTime()

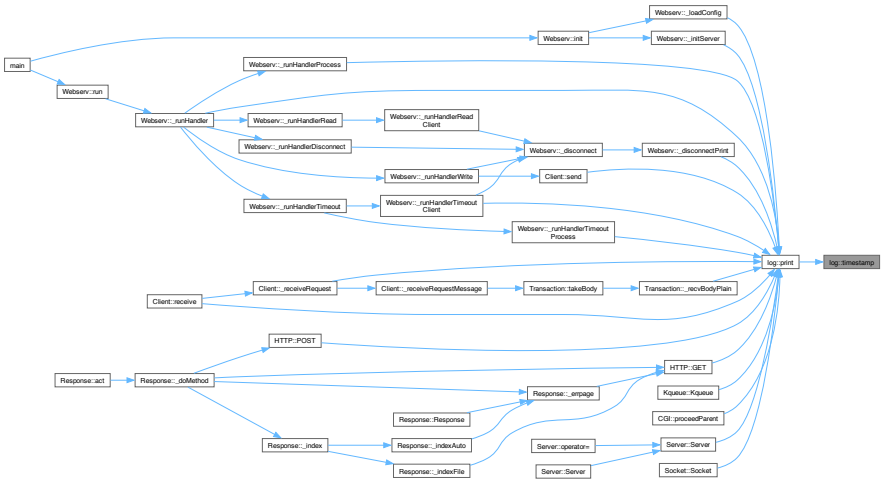
```
std::string log::strTime (
            void )
```

Here is the caller graph for this function:

5.1.1.5 timestamp()

```
void log::timestamp (
            void )
```

Here is the caller graph for this function:



5.1.2 Variable Documentation

5.1.2.1 begin

```
const std::time_t log::begin = std::time( NULL )  [extern]
```

5.1.2.2 history

```
File log::history  [extern]
```

5.2 timeout Namespace Reference

Functions

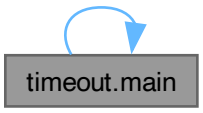
- main ()

5.2.1 Function Documentation

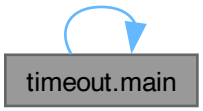
5.2.1.1 main()

```
timeout.main ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



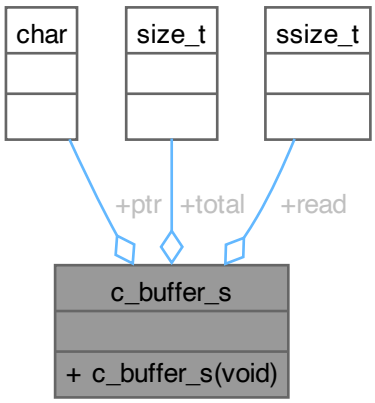
Chapter 6

Class Documentation

6.1 c_buffer_s Struct Reference

`#include <CGI.hpp>`

Collaboration diagram for c_buffer_s:



Public Member Functions

- c_buffer_s (void)

Public Attributes

- char ptr [SIZE_BUFF_C]
- size_t total
- ssize_t read

6.1.1 Constructor & Destructor Documentation

6.1.1.1 c_buffer_s()

```
c_buffer_s::c_buffer_s (
            void )
```

6.1.2 Member Data Documentation

6.1.2.1 ptr

```
char c_buffer_s::ptr[SIZE_BUFF_C]
```

6.1.2.2 read

```
ssize_t c_buffer_s::read
```

6.1.2.3 total

```
size_t c_buffer_s::total
```

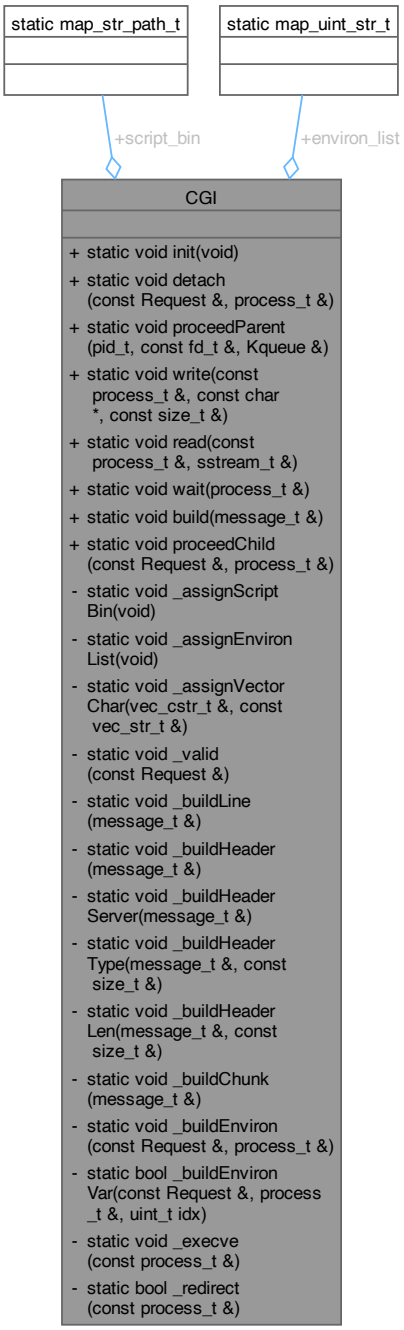
The documentation for this struct was generated from the following files:

- src/http/module/CGI.hpp
- src/http/module/CGI.cpp

6.2 CGI Class Reference

```
#include <CGI.hpp>
```

Collaboration diagram for CGI:



Static Public Member Functions

- static void init (void)
- static void detach (const Request &, process_t &)
- static void proceedParent (pid_t, const fd_t &, Kqueue &)
- static void write (const process_t &, const char *, const size_t &)
- static void read (const process_t &, sstream_t &)

- static void wait (process_t &)
- static void build (message_t &)
- static void proceedChild (const Request &, process_t &)

Static Public Attributes

- static map_str_path_t script_bin
- static map_uint_str_t environ_list

Static Private Member Functions

- static void _assignScriptBin (void)
- static void _assignEnvironList (void)
- static void _assignVectorChar (vec_cstr_t &, const vec_str_t &)
- static void _valid (const Request &)
- static void _buildLine (message_t &)
- static void _buildHeader (message_t &)
- static void _buildHeaderServer (message_t &)
- static void _buildHeaderType (message_t &, const size_t &)
- static void _buildHeaderLen (message_t &, const size_t &)
- static void _buildChunk (message_t &)
- static void _buildEnviron (const Request &, process_t &)
- static bool _buildEnvironVar (const Request &, process_t &, uint_t idx)
- static void _execve (const process_t &)
- static bool _redirect (const process_t &)

6.2.1 Member Function Documentation

6.2.1.1 _assignEnvironList()

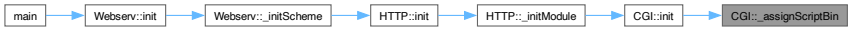
```
void CGI::_assignEnvironList (
            void )  [static], [private]
```

Here is the caller graph for this function:

6.2.1.2 _assignScriptBin()

```
void CGI::_assignScriptBin (
            void )  [static], [private]
```

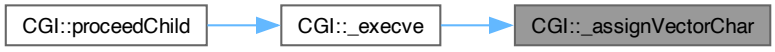
Here is the caller graph for this function:



6.2.1.3 _assignVectorChar()

```
void CGI::_assignVectorChar (
            vec_cstr_t & vec_char,
            const vec_str_t & vec_str)  [static], [private]
```

Here is the caller graph for this function:



6.2.1.4 _buildChunk()

```
void CGI::_buildChunk (
            message_t & out)  [static], [private]
```

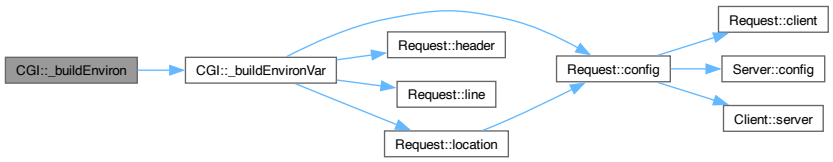
Here is the call graph for this function:

Here is the caller graph for this function:

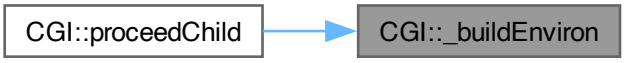
6.2.1.5 _buildEnviron()

```
void CGI::_buildEnviron (
            const Request & rqst,
            process_t & procs)  [static], [private]
```

Here is the call graph for this function:



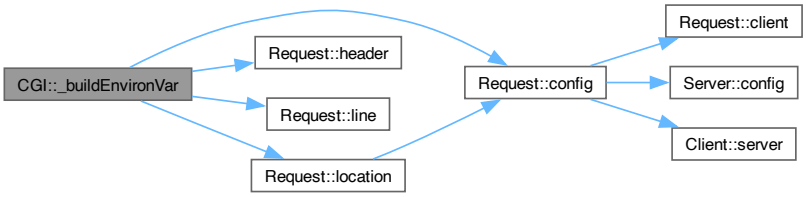
Here is the caller graph for this function:



6.2.1.6 _buildEnvironVar()

```
bool CGI::_buildEnvironVar (
            const Request & rqst,
            process_t & procs,
            uint_t idx)  [static], [private]
```

Here is the call graph for this function:

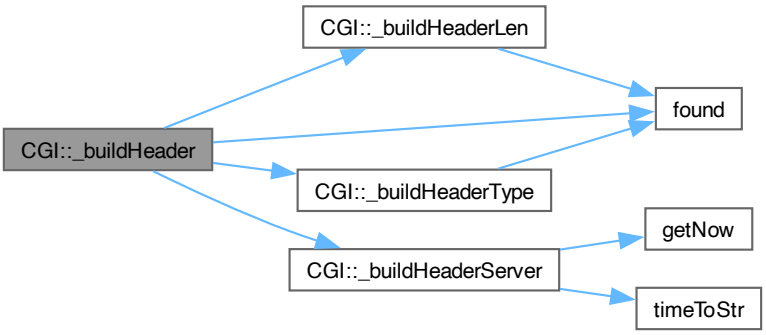


Here is the caller graph for this function:

6.2.1.7 _buildHeader()

```
void CGI::_buildHeader (
            message_t & out)  [static], [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.2.1.8 _buildHeaderLen()

```
void CGI::_buildHeaderLen (
            message_t & out,
            const size_t & pos_header_end)  [static], [private]
```

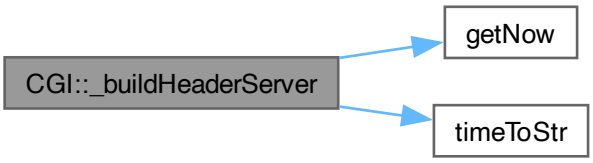
Here is the call graph for this function:

Here is the caller graph for this function:

6.2.1.9 _buildHeaderServer()

```
void CGI::_buildHeaderServer (
            message_t & out)  [static], [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.2.1.10 _buildHeaderType()

```
void CGI::_buildHeaderType (
            message_t & out,
            const size_t & pos_header_end)  [static], [private]
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.2.1.11 _buildLine()

```
void CGI::_buildLine (
            message_t & out)  [static], [private]
```

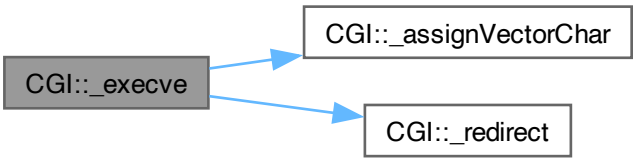
Here is the call graph for this function:

Here is the caller graph for this function:

6.2.1.12 _execve()

```
void CGI::_execve (
            const process_t & procs)  [static], [private]
```

Here is the call graph for this function:

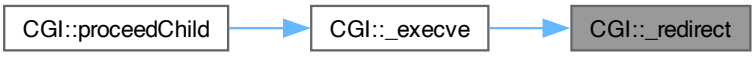


Here is the caller graph for this function:

6.2.1.13 _redirect()

```
bool CGI::_redirect (
            const process_t & procs)  [static], [private]
```

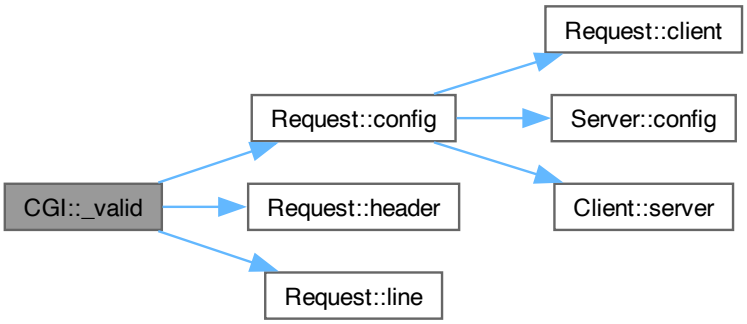
Here is the caller graph for this function:



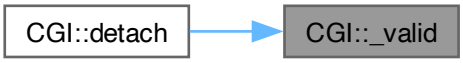
6.2.1.14 _valid()

```
void CGI::_valid (
            const Request & rqst)  [static], [private]
```

Here is the call graph for this function:



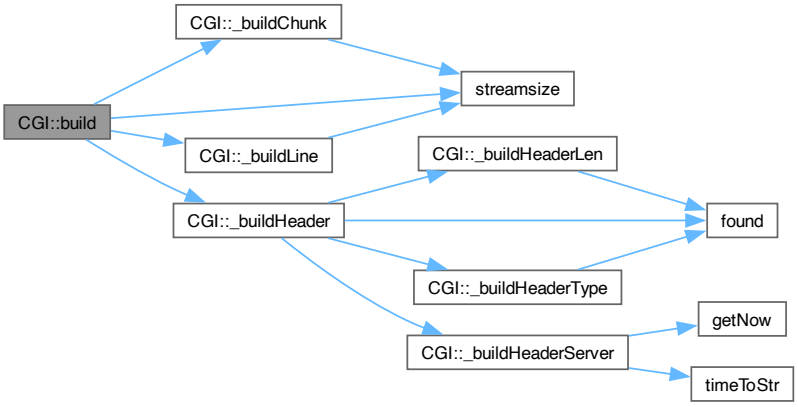
Here is the caller graph for this function:



6.2.1.15 build()

```
void CGI::build (
            message_t & out)  [static]
```

Here is the call graph for this function:

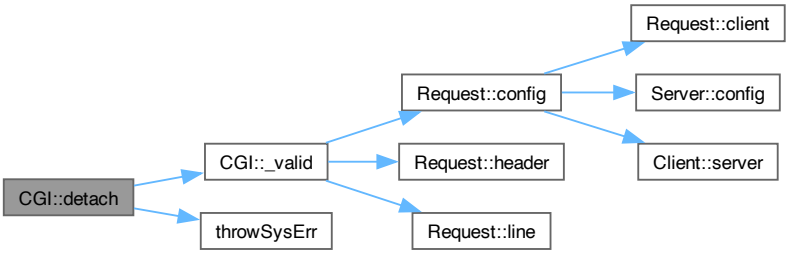


Here is the caller graph for this function:

6.2.1.16 detach()

```
void CGI::detach (
            const Request & rqst,
            process_t & procs)  [static]
```

Here is the call graph for this function:



6.2.1.17 init()

```
void CGI::init (
            void )  [static]
```

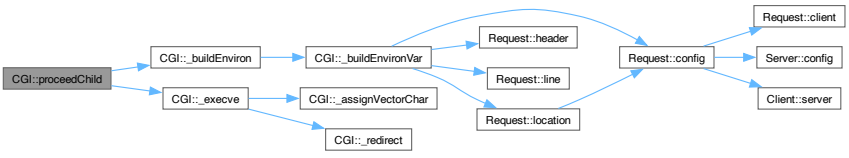
Here is the call graph for this function:

Here is the caller graph for this function:

6.2.1.18 proceedChild()

```
void CGI::proceedChild (
            const Request & rqst,
            process_t & procs)  [static]
```

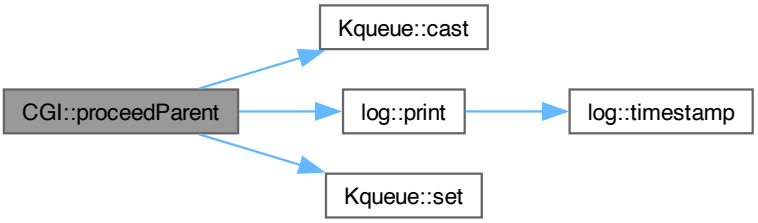
Here is the call graph for this function:



6.2.1.19 proceedParent()

```
void CGI::proceedParent (
            pid_t pid,
            const fd_t & sock_cl,
            Kqueue & evnt)  [static]
```

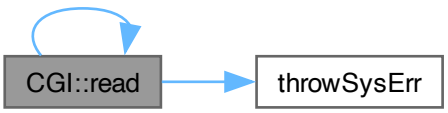
Here is the call graph for this function:



6.2.1.20 read()

```
void CGI::read (
            const process_t & procs,
            sstream_t & out_body)  [static]
```

Here is the call graph for this function:

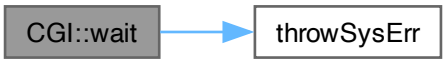


Here is the caller graph for this function:

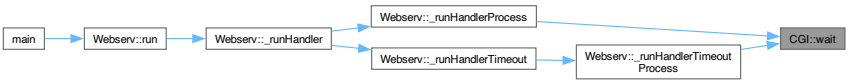
6.2.1.21 wait()

```
void CGI::wait (
            process_t & procs)  [static]
```

Here is the call graph for this function:



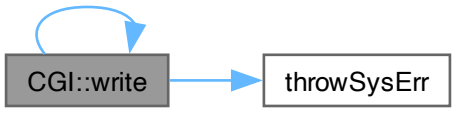
Here is the caller graph for this function:



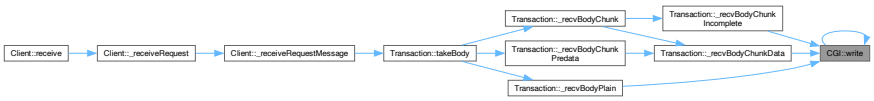
6.2.1.22 write()

```
void CGI::write (
            const process_t & procs,
            const char * in_body,
            const size_t & size)  [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.2 Member Data Documentation

6.2.2.1 environ_list

```
map_uint_str_t CGI::environ_list  [static]
```

6.2.2.2 script_bin

```
map_str_path_t CGI::script_bin  [static]
```

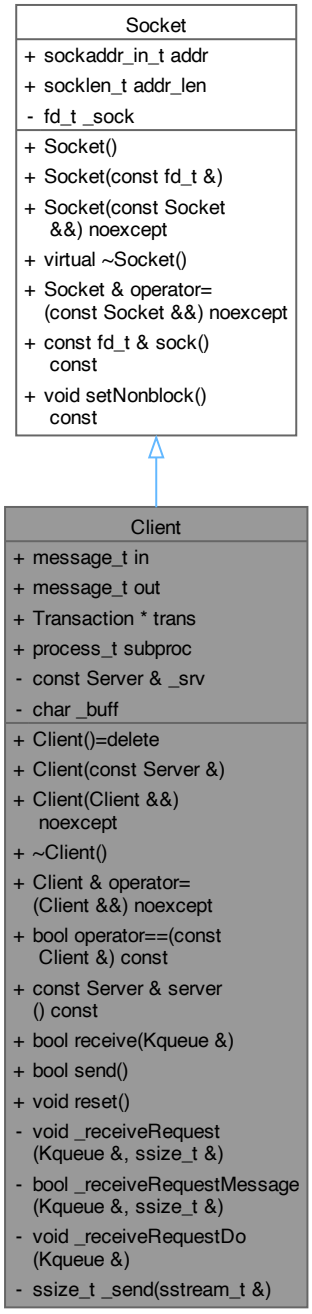
The documentation for this class was generated from the following files:

- src/http/module/CGI.hpp
- src/http/module/CGI.cpp

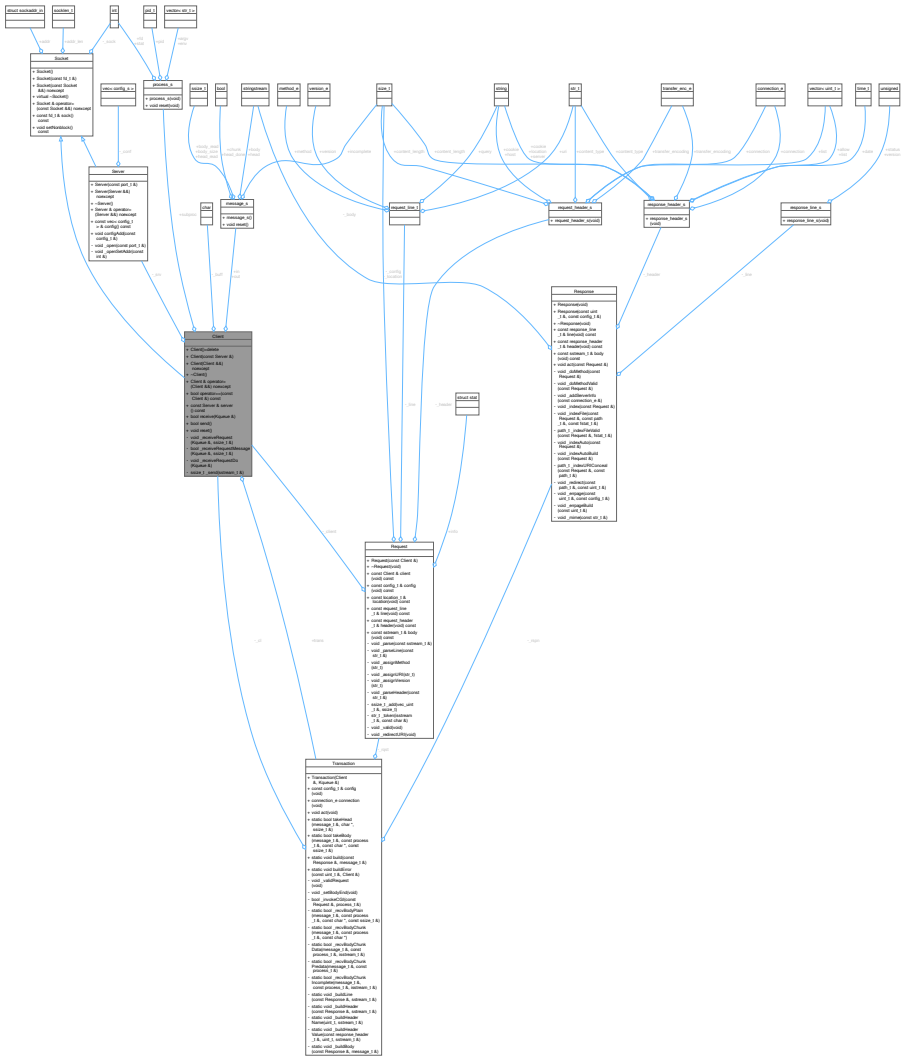
6.3 Client Class Reference

```
#include <Client.hpp>
```

Inheritance diagram for Client:



Collaboration diagram for Client:



Public Member Functions

- Client ()=delete
- Client (const Server &)
- Client (Client &&) noexcept
- ~Client ()
- Client & operator= (Client &&) noexcept
- bool operator== (const Client &) const
- const Server & server () const
- bool receive (Kqueue &)
- bool send ()
- void reset ()

Public Member Functions inherited from Socket

- Socket ()
- Socket (const fd_t &)

- Socket (const Socket &&) noexcept
- virtual ~Socket ()
- Socket & operator= (const Socket &&) noexcept
- const fd_t & sock () const
- void setNonblock () const

Public Attributes

- message_t in
- message_t out
- Transaction * trans
- process_t subproc

Public Attributes inherited from Socket

- sockaddr_in_t addr
- socklen_t addr_len

Private Member Functions

- void _receiveRequest (Kqueue &, ssize_t &)
- bool _receiveRequestMessage (Kqueue &, ssize_t &)
- void _receiveRequestDo (Kqueue &)
- ssize_t _send (sstream_t &)

Private Attributes

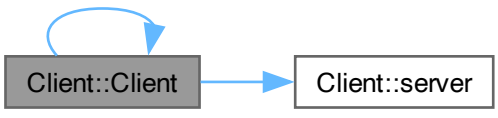
- const Server & _srv
- char _buff [SIZE_BUFF_RECV]

6.3.1 Constructor & Destructor Documentation

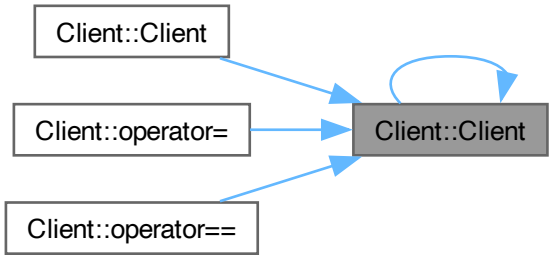
6.3.1.1 Client() [1/3]

```
Client::Client () [delete]
```

Here is the call graph for this function:



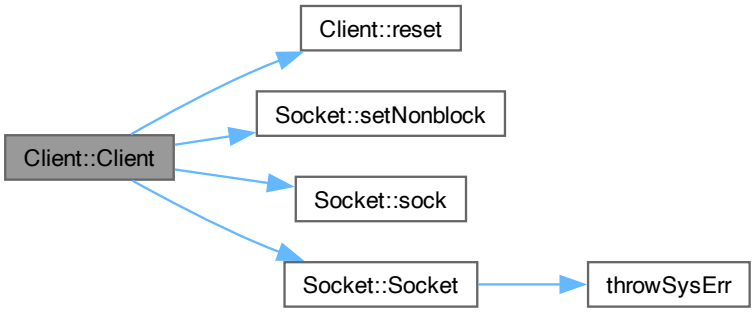
Here is the caller graph for this function:



6.3.1.2 Client() [2/3]

```
Client::Client (
            const Server & srv)
```

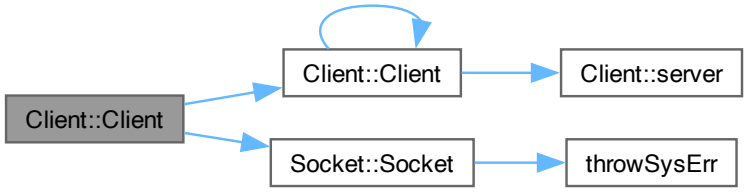
Here is the call graph for this function:



6.3.1.3 Client() [3/3]

```
Client::Client (
            Client && source)  [noexcept]
```

Here is the call graph for this function:



6.3.1.4 ~Client()

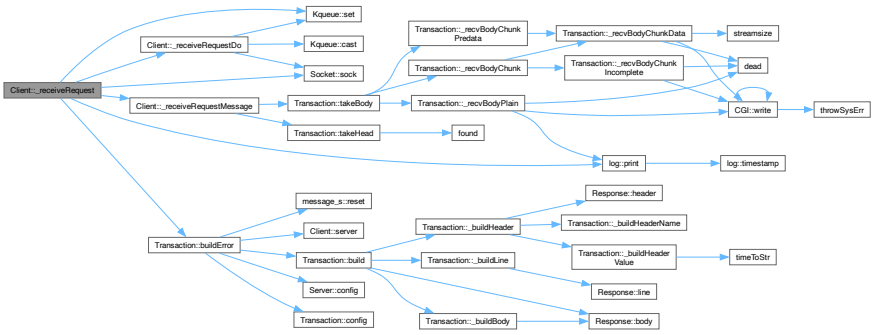
```
Client::~Client ()
```

6.3.2 Member Function Documentation

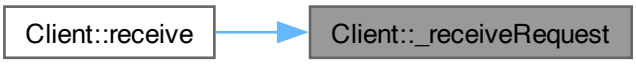
6.3.2.1 _receiveRequest()

```
void Client::_receiveRequest (
            Kqueue & evnt,
            ssize_t & byte_recv)  [private]
```

Here is the call graph for this function:



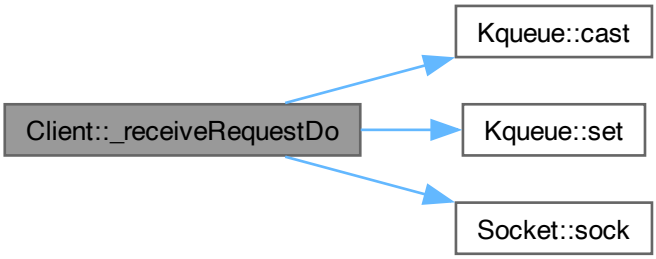
Here is the caller graph for this function:



6.3.2.2 _receiveRequestDo()

```
void Client::_receiveRequestDo (
            Kqueue & evnt)  [private]
```

Here is the call graph for this function:

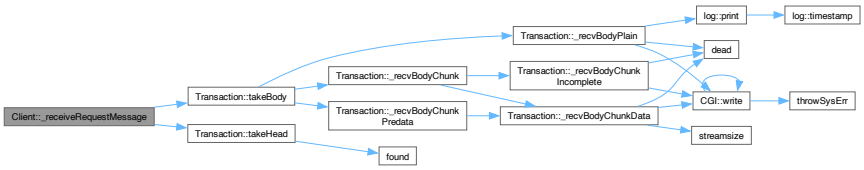


Here is the caller graph for this function:

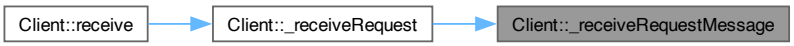
6.3.2.3 _receiveRequestMessage()

```
bool Client::_receiveRequestMessage (
            Kqueue & evnt,
            ssize_t & byte_recv)  [private]
```

Here is the call graph for this function:



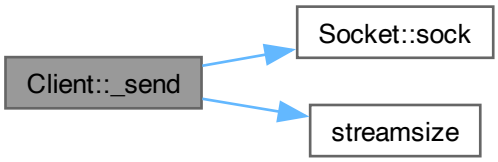
Here is the caller graph for this function:



6.3.2.4 _send()

```
ssize_t Client::_send (
            sstream_t & source)  [private]
```

Here is the call graph for this function:

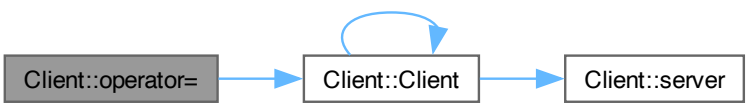


Here is the caller graph for this function:

6.3.2.5 operator=()

```
Client & Client::operator= (
            Client && source)  [noexcept]
```

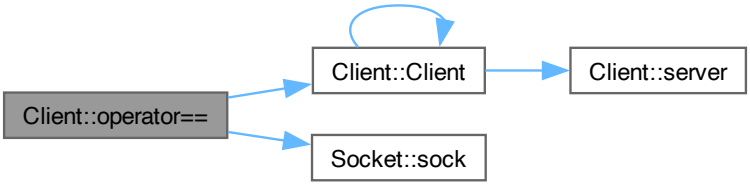
Here is the call graph for this function:



6.3.2.6 operator==()

```
bool Client::operator== (
            const Client & source) const
```

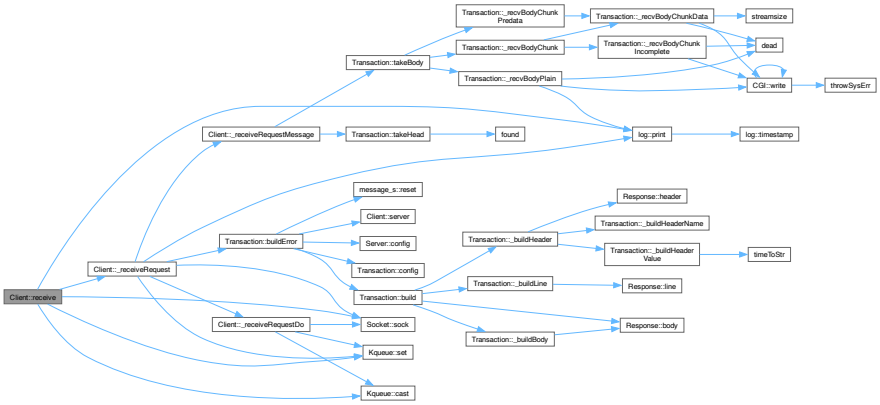
Here is the call graph for this function:



6.3.2.7 receive()

```
bool Client::receive (
            Kqueue & evnt)
```

Here is the call graph for this function:



6.3.2.8 reset()

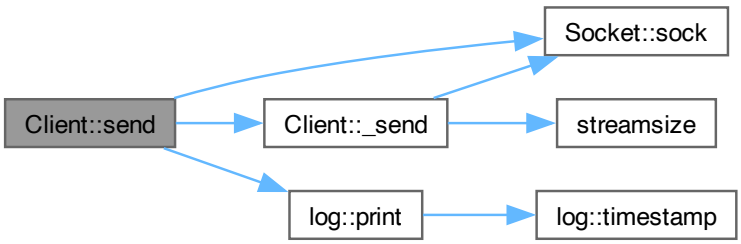
```
void Client::reset ()
```

Here is the caller graph for this function:

6.3.2.9 send()

```
bool Client::send ()
```

Here is the call graph for this function:

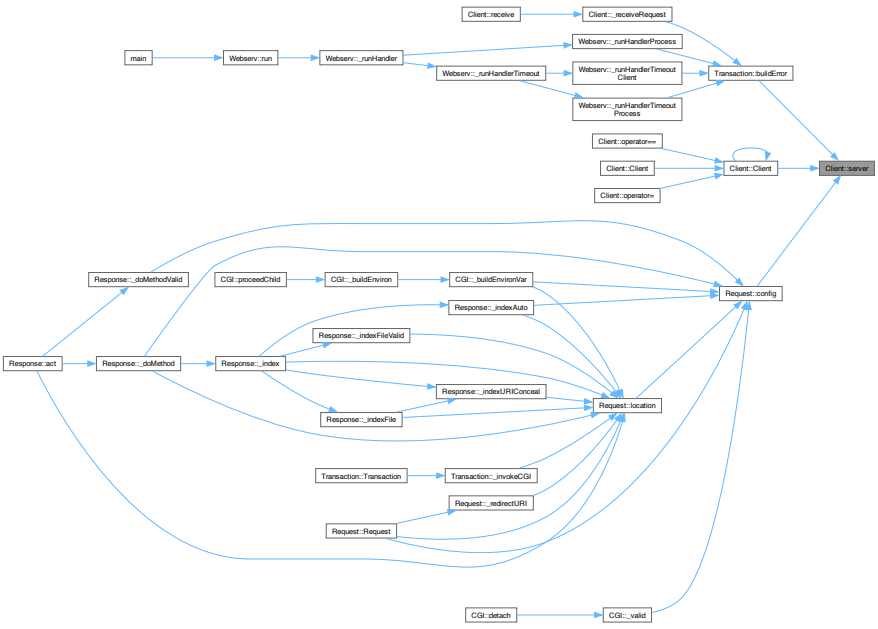


Here is the caller graph for this function:

6.3.2.10 server()

```
const Server & Client::server () const
```

Here is the caller graph for this function:



6.3.3 Member Data Documentation

6.3.3.1 _buff

```
char Client::_buff[SIZE_BUFF_RECV]  [private]
```

6.3.3.2 _srv

```
const Server& Client::_srv  [private]
```

6.3.3.3 in

```
message_t Client::in
```

6.3.3.4 out

```
message_t Client::out
```

6.3.3.5 subproc

```
process_t Client::subproc
```

6.3.3.6 trans

```
Transaction* Client::trans
```

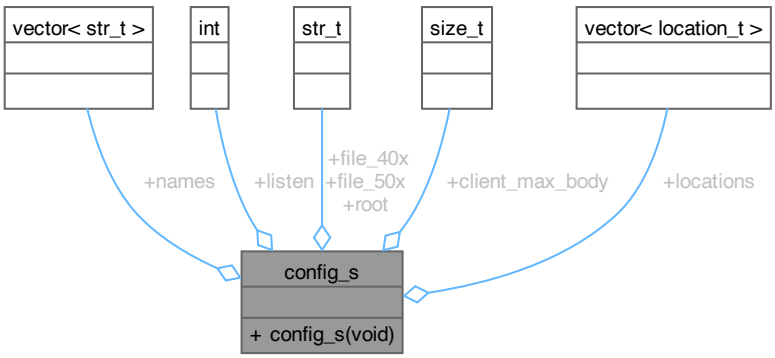
The documentation for this class was generated from the following files:

- src/core/Client.hpp
- src/core/Client.cpp

6.4 config_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for config_s:



Public Member Functions

- config_s (void)

Public Attributes

- vec_str_t names
- port_t listen
- path_t root
- name_t file_40x
- name_t file_50x
- size_t client_max_body
- vec_location_t locations

6.4.1 Constructor & Destructor Documentation

6.4.1.1 config_s()

```
config_s::config_s (
            void )
```

6.4.2 Member Data Documentation

6.4.2.1 client_max_body

```
size_t config_s::client_max_body
```

6.4.2.2 file_40x

```
name_t config_s::file_40x
```

6.4.2.3 file_50x

```
name_t config_s::file_50x
```

6.4.2.4 listen

```
port_t config_s::listen
```

6.4.2.5 locations

```
vec_location_t config_s::locations
```

6.4.2.6 names

```
vec_str_t config_s::names
```

6.4.2.7 root

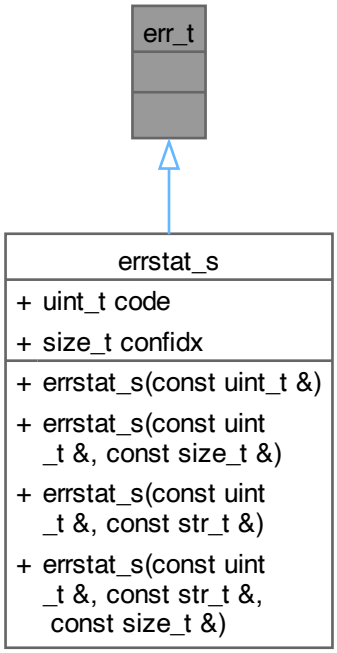
```
path_t config_s::root
```

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

- src/http/filter.hpp
- src/http/HTTP.cpp

6.5 err_t Class Reference

Inheritance diagram for err_t:



Collaboration diagram for err_t:

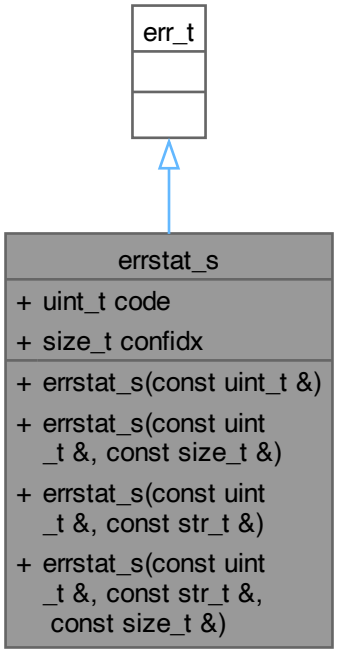
The documentation for this class was generated from the following file:

- src/common/error.hpp

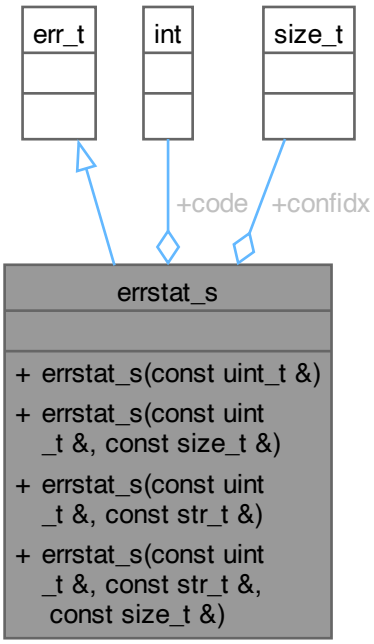
6.6 errstat_s Struct Reference

`#include <error.hpp>`

Inheritance diagram for errstat_s:



Collaboration diagram for errstat_s:



Public Member Functions

- errstat_s (const uint_t &)
- errstat_s (const uint_t &, const size_t &)
- errstat_s (const uint_t &, const str_t &)
- errstat_s (const uint_t &, const str_t &, const size_t &)

Public Attributes

- uint_t code
- size_t confidx

6.6.1 Constructor & Destructor Documentation

6.6.1.1 errstat_s() [1/4]

```
errstat_s::errstat_s (
            const uint_t & status)
```

6.6.1.2 errstat_s() [2/4]

```
errstat_s::errstat_s (
            const uint_t & status,
            const size_t & idx)
```

6.6.1.3 errstat_s() [3/4]

```
errstat_s::errstat_s (
            const uint_t & status,
            const str_t & msg)
```

6.6.1.4 errstat_s() [4/4]

```
errstat_s::errstat_s (
            const uint_t & status,
            const str_t & msg,
            const size_t & idx)
```

6.6.2 Member Data Documentation

6.6.2.1 code

```
uint_t errstat_s::code
```

6.6.2.2 confidx

```
size_t errstat_s::confidx
```

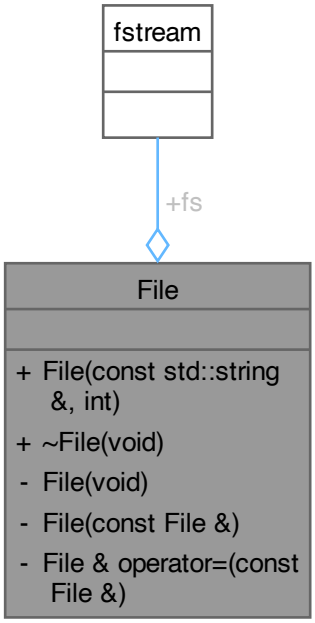
The documentation for this struct was generated from the following files:

- src/common/error.hpp
- src/common/error.cpp

6.7 File Class Reference

```
#include <File.hpp>
```

Collaboration diagram for File:



Public Member Functions

- File (const std::string &, int)
- ~File (void)

Public Attributes

- std::fstream fs

Private Member Functions

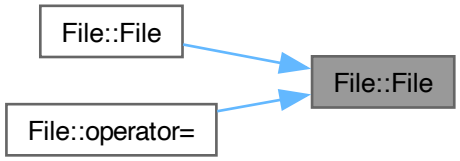
- File (void)
- File (const File &)
- File & operator= (const File &)

6.7.1 Constructor & Destructor Documentation

6.7.1.1 File() [1/3]

```
File::File (
            const std::string & fileName,
            int mode)
```

Here is the caller graph for this function:



6.7.1.2 ~File()

```
File::~File (
            void )
```

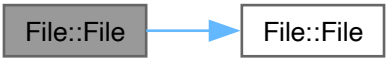
6.7.1.3 File() [2/3]

```
File::File (
            void )  [private]
```

6.7.1.4 File() [3/3]

```
File::File (
            const File & )  [private]
```

Here is the call graph for this function:



6.7.2 Member Function Documentation

6.7.2.1 operator=()

```
File & File::operator= (
            const File & )  [private]
```

Here is the call graph for this function:

6.7.3 Member Data Documentation

6.7.3.1 fs

```
std::fstream File::fs
```

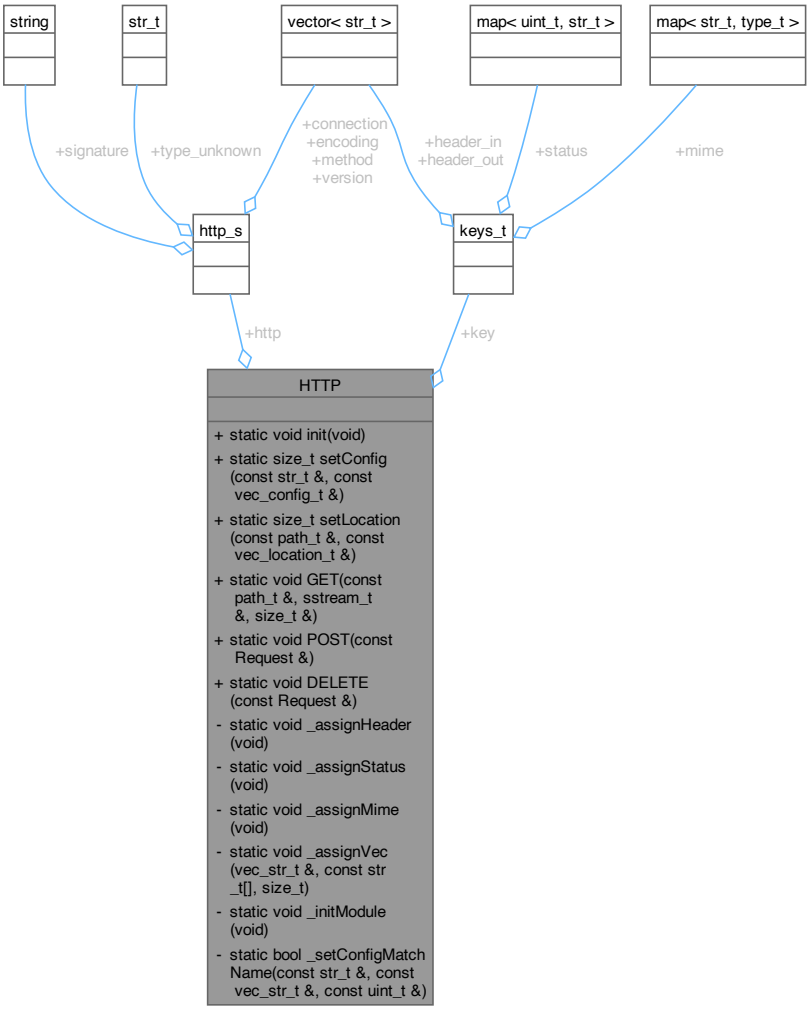
The documentation for this class was generated from the following files:

- src/common/File.hpp
- src/common/File.cpp

6.8 HTTP Class Reference

```
#include <HTTP.hpp>
```

Collaboration diagram for HTTP:



Static Public Member Functions

- static void init (void)
- static size_t setConfig (const str_t &, const vec_config_t &)
- static size_t setLocation (const path_t &, const vec_location_t &)
- static void GET (const path_t &, sstream_t &, size_t &)
- static void POST (const Request &)
- static void DELETE (const Request &)

Static Public Attributes

- static http_t http
- static keys_t key

Static Private Member Functions

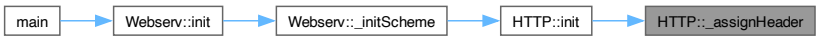
- static void _assignHeader (void)
- static void _assignStatus (void)
- static void _assignMime (void)
- static void _assignVec (vec_str_t &, const str_t[ ], size_t)
- static void _initModule (void)
- static bool _setConfigMatchName (const str_t &, const vec_str_t &, const uint_t &)

6.8.1 Member Function Documentation

6.8.1.1 _assignHeader()

```
void HTTP::_assignHeader (
            void )  [static], [private]
```

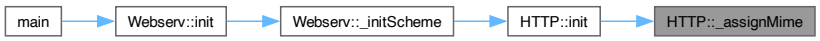
Here is the caller graph for this function:



6.8.1.2 _assignMime()

```
void HTTP::_assignMime (
            void )  [static], [private]
```

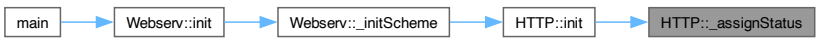
Here is the caller graph for this function:



6.8.1.3 _assignStatus()

```
void HTTP::_assignStatus (
            void )  [static], [private]
```

Here is the caller graph for this function:



6.8.1.4 _assignVec()

```
void HTTP::_assignVec (
            vec_str_t & target,
            const str_t source[],
            size_t cnt)  [static], [private]
```

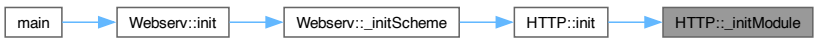
Here is the caller graph for this function:

6.8.1.5 _initModule()

```
void HTTP::_initModule (
            void )  [static], [private]
```

Here is the call graph for this function:

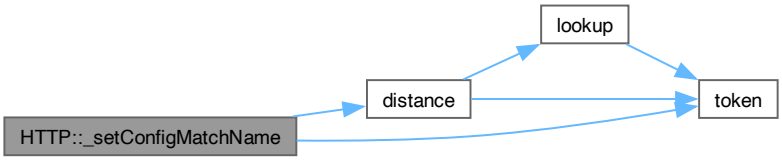
Here is the caller graph for this function:



6.8.1.6 _setConfigMatchName()

```
bool HTTP::_setConfigMatchName (
            const str_t & host,
            const vec_str_t & names,
            const uint_t & listen)  [static], [private]
```

Here is the call graph for this function:

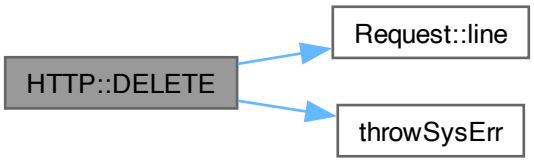


Here is the caller graph for this function:

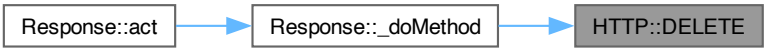
6.8.1.7 DELETE()

```
void HTTP::DELETE (
            const Request & rqst)  [static]
```

Here is the call graph for this function:



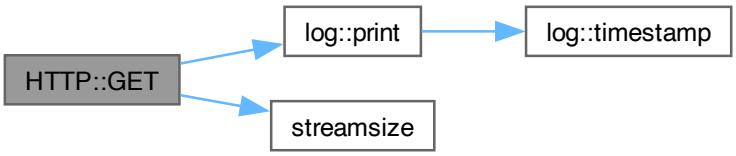
Here is the caller graph for this function:



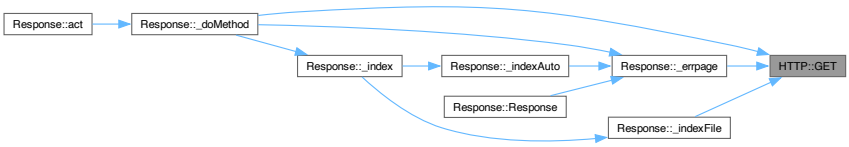
6.8.1.8 GET()

```
void HTTP::GET (
            const path_t & uri,
            sstream_t & body,
            size_t & size)  [static]
```

Here is the call graph for this function:



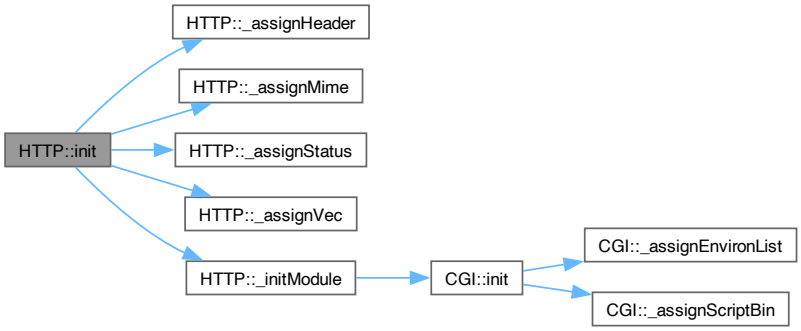
Here is the caller graph for this function:



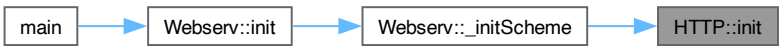
6.8.1.9 init()

```
void HTTP::init (
            void )  [static]
```

Here is the call graph for this function:



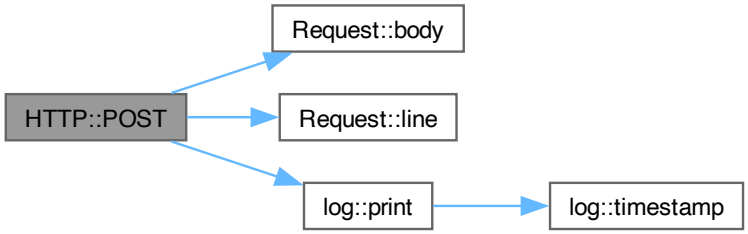
Here is the caller graph for this function:



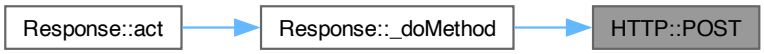
6.8.1.10 POST()

```
void HTTP::POST (
            const Request & rqst)  [static]
```

Here is the call graph for this function:



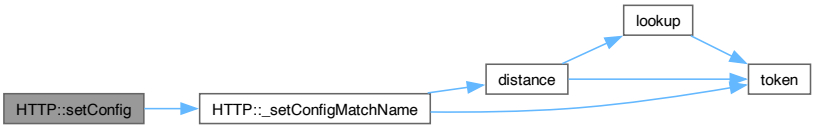
Here is the caller graph for this function:



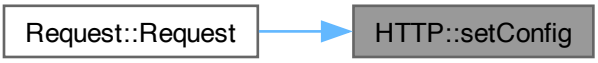
6.8.1.11 setConfig()

```
size_t HTTP::setConfig (
            const str_t & host,
            const vec_config_t & configs)  [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

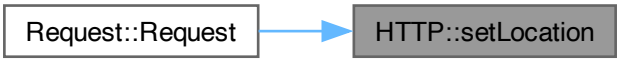


6.8.1.12 setLocation()

```
size_t HTTP::setLocation (
            const path_t & uri,
            const vec_location_t & locations)  [static]
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.8.2 Member Data Documentation

6.8.2.1 http

```
http_t HTTP::http  [static]
```

6.8.2.2 key

```
keys_t HTTP::key  [static]
```

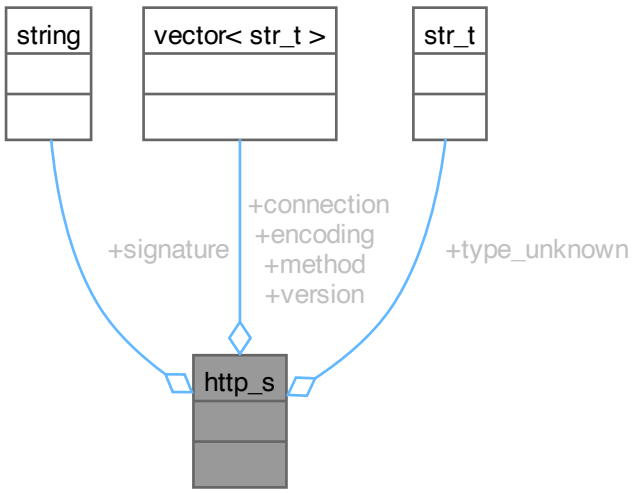
The documentation for this class was generated from the following files:

- src/http/HTTP.hpp
- src/http/HTTP.cpp

6.9 http_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for http_s:



Public Attributes

- str_t signature
- vec_str_t version
- vec_str_t method
- type_t type_unknown
- vec_str_t encoding
- vec_str_t connection

6.9.1 Member Data Documentation

6.9.1.1 connection

```
vec_str_t http_s::connection
```

6.9.1.2 encoding

```
vec_str_t http_s::encoding
```

6.9.1.3 method

```
vec_str_t http_s::method
```

6.9.1.4 signature

```
str_t http_s::signature
```

6.9.1.5 type_unknown

```
type_t http_s::type_unknown
```

6.9.1.6 version

```
vec_str_t http_s::version
```

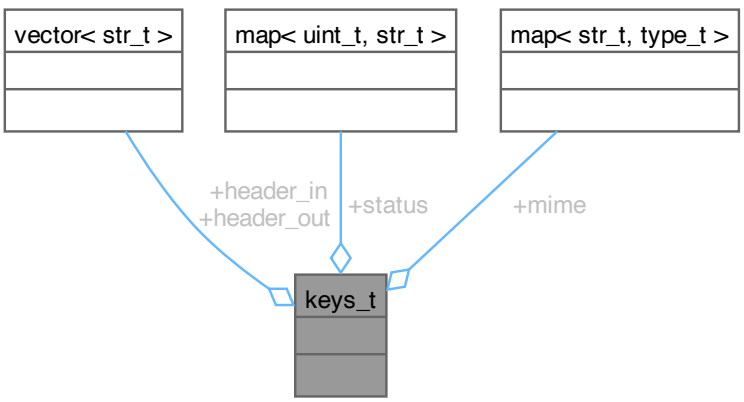
The documentation for this struct was generated from the following file:

- src/http/filter.hpp

6.10 keys_t Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for keys_t:



Public Attributes

- vec_str_t header_in
- vec_str_t header_out
- map_uint_str_t status
- map_str_type_t mime

6.10.1 Member Data Documentation

6.10.1.1 header_in

```
vec_str_t keys_t::header_in
```

6.10.1.2 header_out

```
vec_str_t keys_t::header_out
```

6.10.1.3 mime

```
map_str_type_t keys_t::mime
```

6.10.1.4 status

```
map_uint_str_t keys_t::status
```

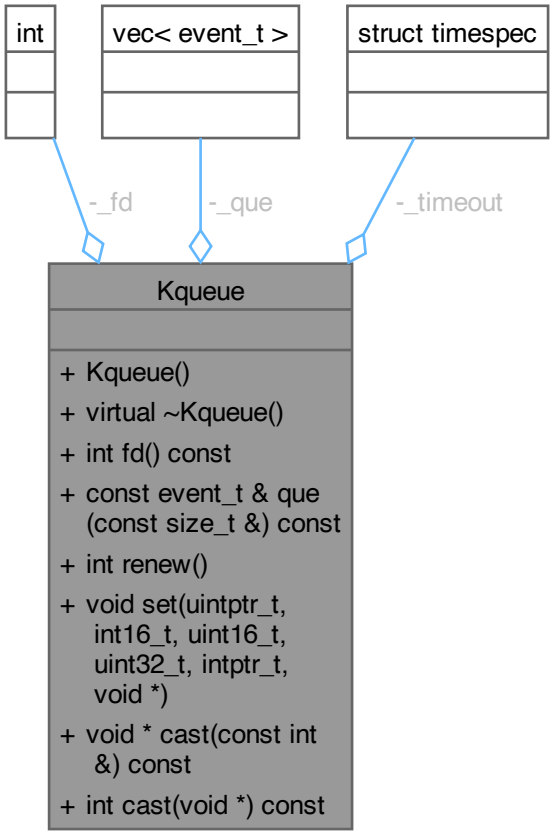
The documentation for this struct was generated from the following file:

- src/http/filter.hpp

6.11 Kqueue Class Reference

```
#include <Kqueue.hpp>
```

Collaboration diagram for Kqueue:



Public Member Functions

- Kqueue ()
- virtual ~Kqueue ()
- int fd () const
- const event_t & que (const size_t &) const
- int renew ()
- void set (uintptr_t, int16_t, uint16_t, uint32_t, intptr_t, void *)
- void * cast (const int &) const
- int cast (void *) const

Private Attributes

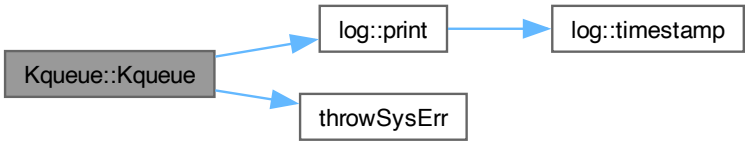
- int _fd
- vec< event_t > _que
- struct timespec _timeout

6.11.1 Constructor & Destructor Documentation

6.11.1.1 Kqueue()

```
Kqueue::Kqueue ()
```

Here is the call graph for this function:



6.11.1.2 ~Kqueue()

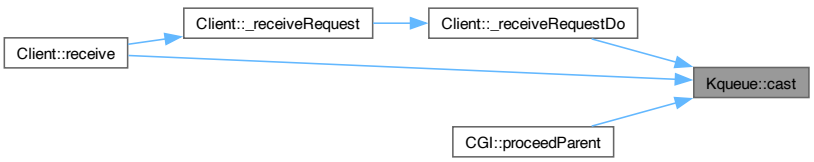
```
Kqueue::~Kqueue ()  [virtual]
```

6.11.2 Member Function Documentation

6.11.2.1 cast() [1/2]

```
void * Kqueue::cast (
            const int & target) const
```

Here is the caller graph for this function:



6.11.2.2 cast() [2/2]

```
int Kqueue::cast (
            void * target) const
```

6.11.2.3 fd()

```
int Kqueue::fd () const
```

6.11.2.4 que()

```
const event_t & Kqueue::que (
            const size_t & i) const
```

6.11.2.5 renew()

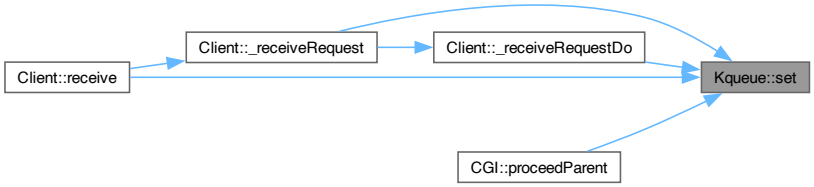
```
int Kqueue::renew ()
```

Here is the call graph for this function:

6.11.2.6 set()

```
void Kqueue::set (
            uintptr_t ident,
            int16_t filter,
            uint16_t flags,
            uint32_t fflags,
            intptr_t data,
            void * udata)
```

Here is the caller graph for this function:



6.11.3 Member Data Documentation

6.11.3.1 _fd

```
int Kqueue::_fd  [private]
```

6.11.3.2 _que

```
vec<event_t> Kqueue::_que  [private]
```

6.11.3.3 _timeout

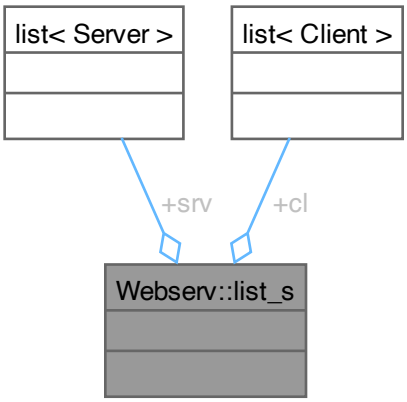
```
struct timespec Kqueue::_timeout  [private]
```

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

- src/common/Kqueue.hpp
- src/common/Kqueue.cpp

6.12 Webserv::list_s Struct Reference

Collaboration diagram for Webserv::list_s:



Public Attributes

- list< Server > srv
- list< Client > cl

6.12.1 Member Data Documentation

6.12.1.1 cl

```
list<Client> Webserv::list_s::cl
```

6.12.1.2 srv

```
list<Server> Webserv::list_s::srv
```

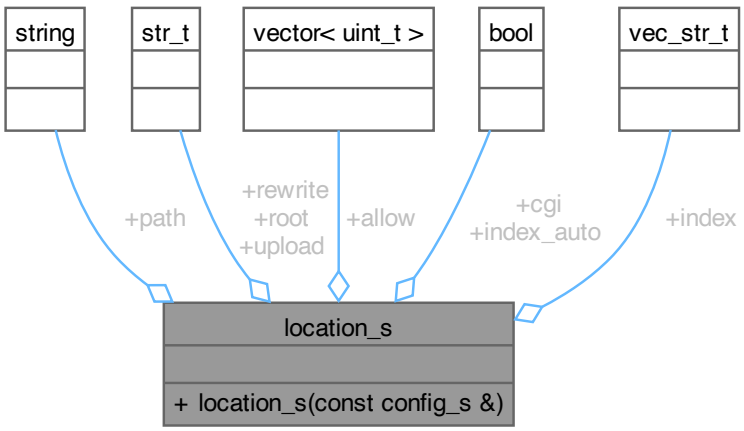
The documentation for this struct was generated from the following file:

- src/core/Webserv.hpp

6.13 location_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for location_s:



Public Member Functions

- location_s (const config_s &)

Public Attributes

- str_t path
- path_t root
- path_t rewrite
- vec_uint_t allow
- bool cgi
- path_t upload
- vec_name_t index
- bool index_auto

6.13.1 Constructor & Destructor Documentation

6.13.1.1 location_s()

```
location_s::location_s (
            const config_s & serverconf)
```

6.13.2 Member Data Documentation

6.13.2.1 allow

```
vec_uint_t location_s::allow
```

6.13.2.2 cgi

```
bool location_s::cgi
```

6.13.2.3 index

```
vec_name_t location_s::index
```

6.13.2.4 index_auto

```
bool location_s::index_auto
```

6.13.2.5 path

```
str_t location_s::path
```

6.13.2.6 rewrite

```
path_t location_s::rewrite
```

6.13.2.7 root

```
path_t location_s::root
```

6.13.2.8 upload

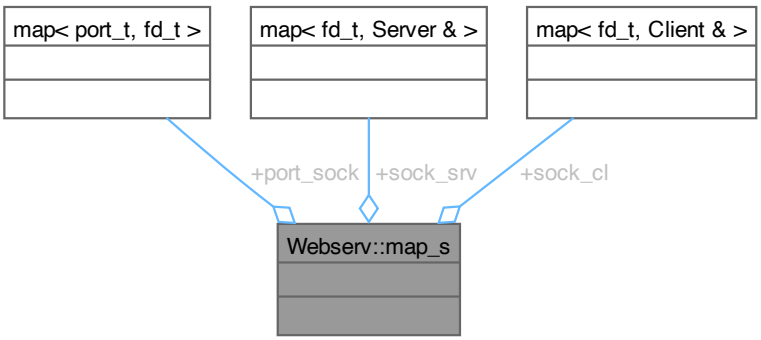
```
path_t location_s::upload
```

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

- src/http/filter.hpp
- src/http/HTTP.cpp

6.14 Webserv::map_s Struct Reference

Collaboration diagram for Webserv::map_s:



Public Attributes

- map< port_t, fd_t > port_sock
- map< fd_t, Server & > sock_srv
- map< fd_t, Client & > sock_cl

6.14.1 Member Data Documentation

6.14.1.1 port_sock

```
map<port_t, fd_t> Webserv::map_s::port_sock
```

6.14.1.2 sock_cl

```
map<fd_t, Client&> Webserv::map_s::sock_cl
```

6.14.1.3 sock_srv

```
map<fd_t, Server&> Webserv::map_s::sock_srv
```

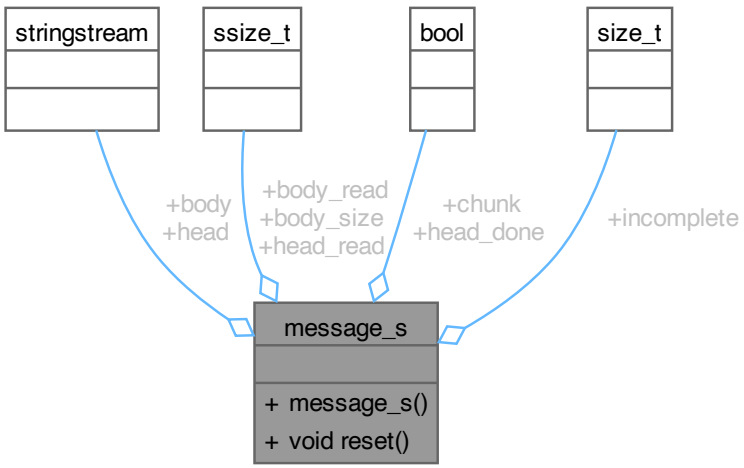
The documentation for this struct was generated from the following file:

- src/core/Webserv.hpp

6.15 message_s Struct Reference

```
#include <Transaction.hpp>
```

Collaboration diagram for message_s:



Public Member Functions

- message_s ()
- void reset ()

Public Attributes

- sstream_t head
- ssize_t head_read
- bool head_done
- sstream_t body
- ssize_t body_read
- ssize_t body_size
- bool chunk
- size_t incomplete

6.15.1 Constructor & Destructor Documentation

6.15.1.1 message_s()

```
message_s::message_s ()
```

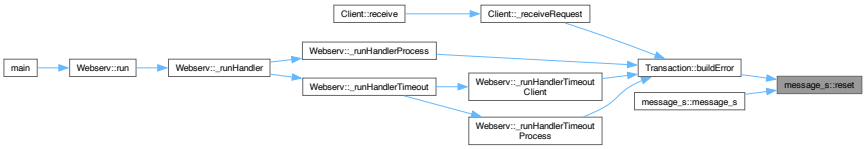
Here is the call graph for this function:

6.15.2 Member Function Documentation

6.15.2.1 reset()

```
void message_s::reset ()
```

Here is the caller graph for this function:



6.15.3 Member Data Documentation

6.15.3.1 body

```
sstream_t message_s::body
```

6.15.3.2 body_read

```
ssize_t message_s::body_read
```

6.15.3.3 body_size

```
ssize_t message_s::body_size
```

6.15.3.4 chunk

```
bool message_s::chunk
```

6.15.3.5 head

```
sstream_t message_s::head
```

6.15.3.6 head_done

```
bool message_s::head_done
```

6.15.3.7 head_read

```
ssize_t message_s::head_read
```

6.15.3.8 incomplete

```
size_t message_s::incomplete
```

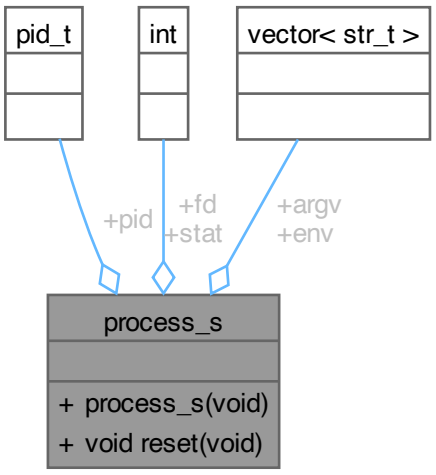
The documentation for this struct was generated from the following files:

- src/http/Transaction.hpp
- src/http/Transaction.cpp

6.16 process_s Struct Reference

```
#include <type.hpp>
```

Collaboration diagram for process_s:



Public Member Functions

- process_s (void)
- void reset (void)

Public Attributes

- pid_t pid
- stat_t stat
- pipe_t fd [2]
- vec_str_t argv
- vec_str_t env

6.16.1 Constructor & Destructor Documentation

6.16.1.1 process_s()

```
process_s::process_s (
            void )
```

Here is the call graph for this function:

6.16.2 Member Function Documentation

6.16.2.1 reset()

```
void process_s::reset (
            void )
```

Here is the caller graph for this function:

6.16.3 Member Data Documentation

6.16.3.1 argv

```
vec_str_t process_s::argv
```

6.16.3.2 env

```
vec_str_t process_s::env
```

6.16.3.3 fd

```
pipe_t process_s::fd[2]
```

6.16.3.4 pid

```
pid_t process_s::pid
```

6.16.3.5 stat

```
stat_t process_s::stat
```

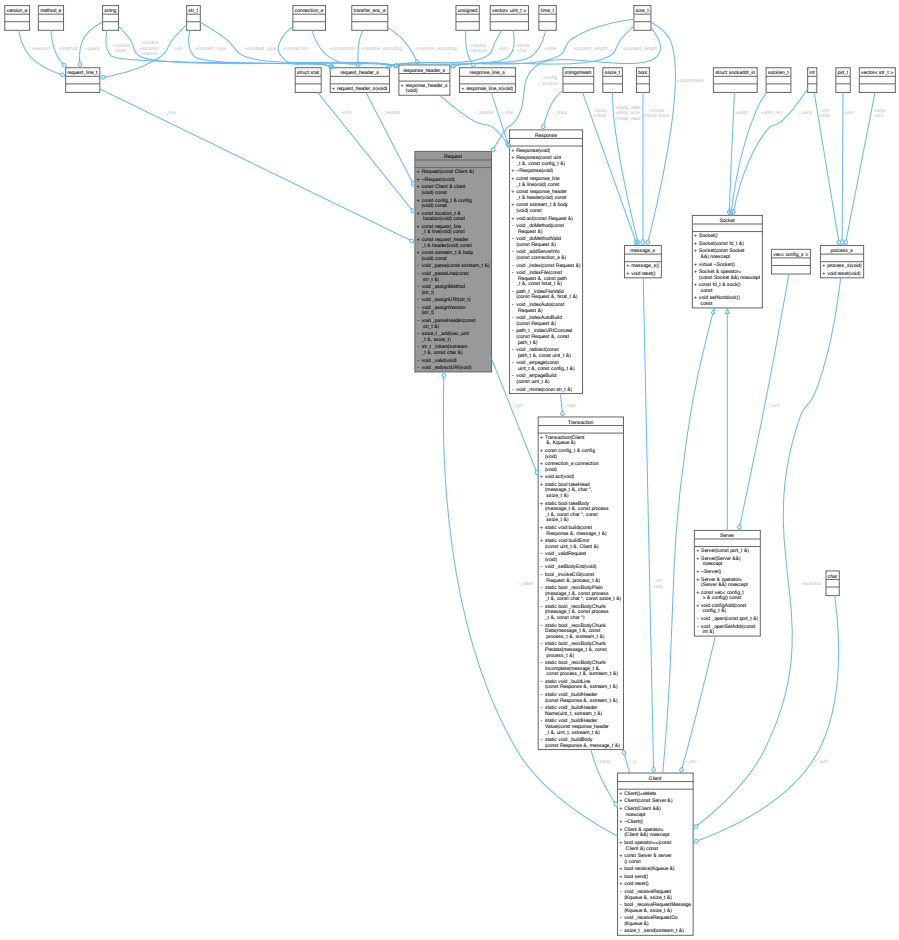
The documentation for this struct was generated from the following files:

- src/common/type.hpp
- src/http/module/CGI.cpp

6.17 Request Class Reference

```
#include <Request.hpp>
```

Collaboration diagram for Request:



Public Member Functions

- Request (const Client &)
- ~Request (void)
- const Client & client (void) const
- const config_t & config (void) const
- const location_t & location (void) const
- const request_line_t & line (void) const
- const request_header_t & header (void) const
- const sstream_t & body (void) const

Public Attributes

- fstat_t info

Private Member Functions

- void _parse (const sstream_t &)
- void _parseLine (const str_t &)
- void _assignMethod (str_t)

- void _assignURI (str_t)
- void _assignVersion (str_t)
- void _parseHeader (const str_t &)
- ssize_t _add (vec_uint_t &, ssize_t)
- str_t _token (isstream_t &, const char &)
- void _valid (void)
- void _redirectURI (void)

Private Attributes

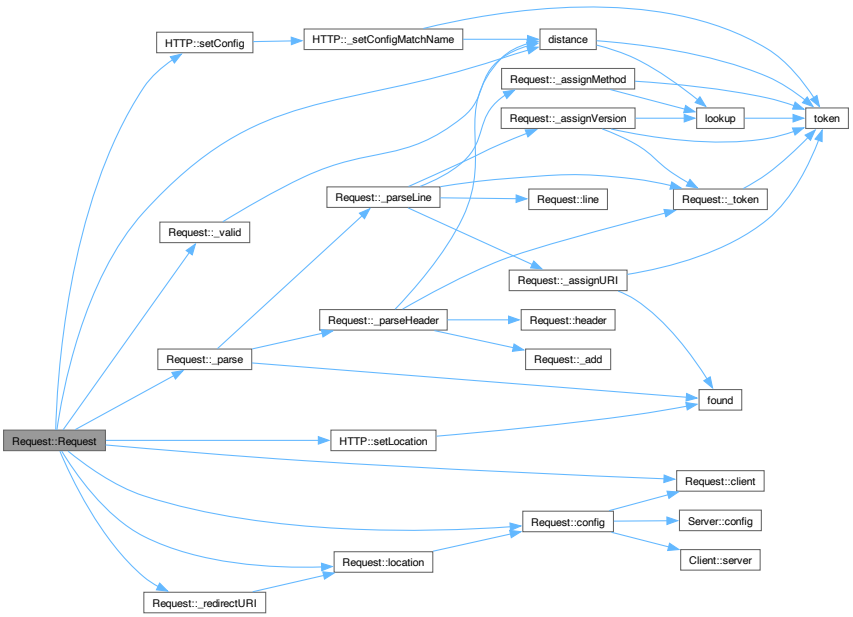
- const Client & _client
- size_t _config
- size_t _location
- request_line_t _line
- request_header_t _header

6.17.1 Constructor & Destructor Documentation

6.17.1.1 Request()

```
Request::Request (
            const Client & client)
```

Here is the call graph for this function:



6.17.1.2 ~Request()

```
Request::~Request (
            void )
```

6.17.2 Member Function Documentation

6.17.2.1 _add()

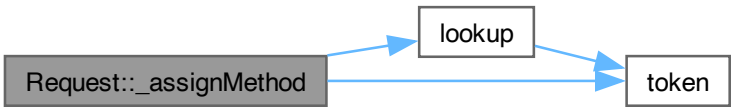
```
ssize_t Request::_add (
            vec_uint_t & list,
            ssize_t id)  [private]
```

Here is the caller graph for this function:

6.17.2.2 _assignMethod()

```
void Request::_assignMethod (
            str_t token)  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.17.2.3 _assignURI()

```
void Request::_assignURI (
            str_t token)  [private]
```

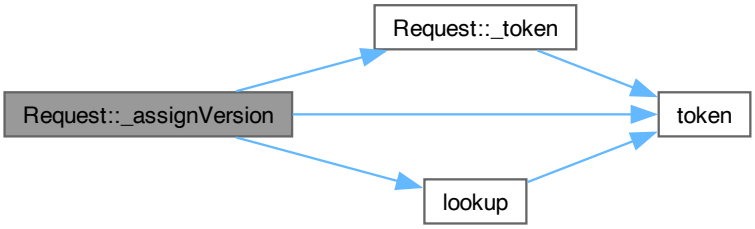
Here is the call graph for this function:

Here is the caller graph for this function:

6.17.2.4 _assignVersion()

```
void Request::_assignVersion (
            str_t token)  [private]
```

Here is the call graph for this function:

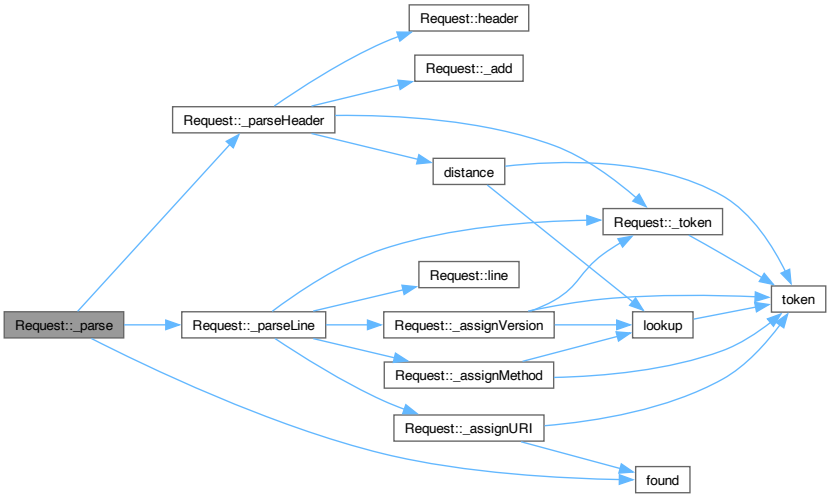


Here is the caller graph for this function:

6.17.2.5 _parse()

```
void Request::_parse (
            const sstream_t & msg)  [private]
```

Here is the call graph for this function:

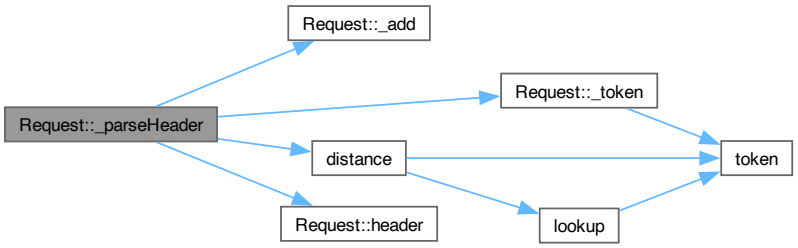


Here is the caller graph for this function:

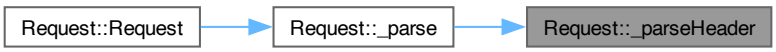
6.17.2.6 _parseHeader()

```
void Request::_parseHeader (
            const str_t & field)  [private]
```

Here is the call graph for this function:



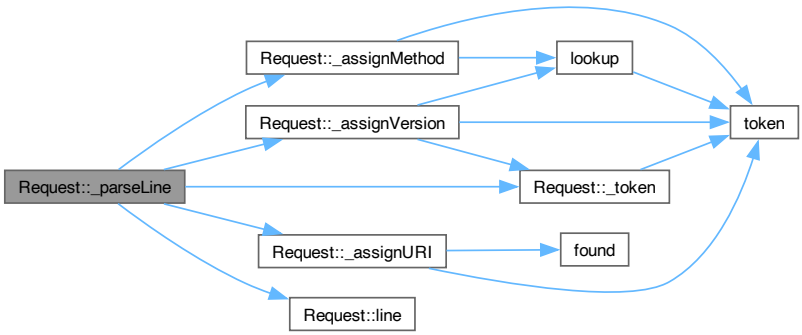
Here is the caller graph for this function:



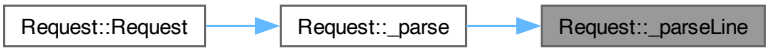
6.17.2.7 _parseLine()

```
void Request::_parseLine (
            const str_t & line)  [private]
```

Here is the call graph for this function:



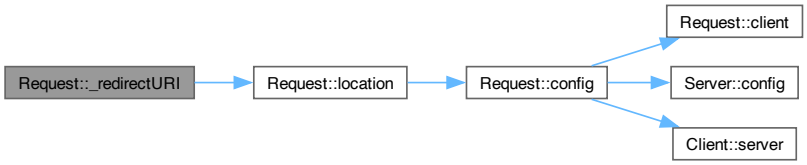
Here is the caller graph for this function:



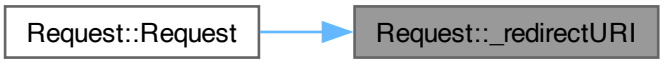
6.17.2.8 _redirectURI()

```
void Request::_redirectURI (
            void )  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

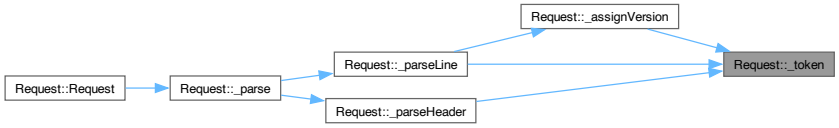


6.17.2.9 _token()

```
str_t Request::_token (
            isstream_t & iss,
            const char & delim)  [private]
```

Here is the call graph for this function:

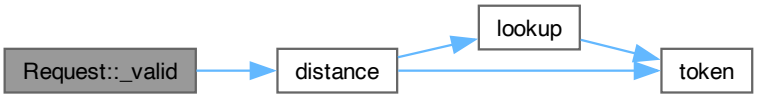
Here is the caller graph for this function:



6.17.2.10 _valid()

```
void Request::_valid (
            void )  [private]
```

Here is the call graph for this function:

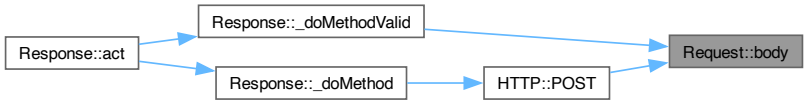


Here is the caller graph for this function:

6.17.2.11 body()

```
const sstream_t & Request::body (
            void ) const
```

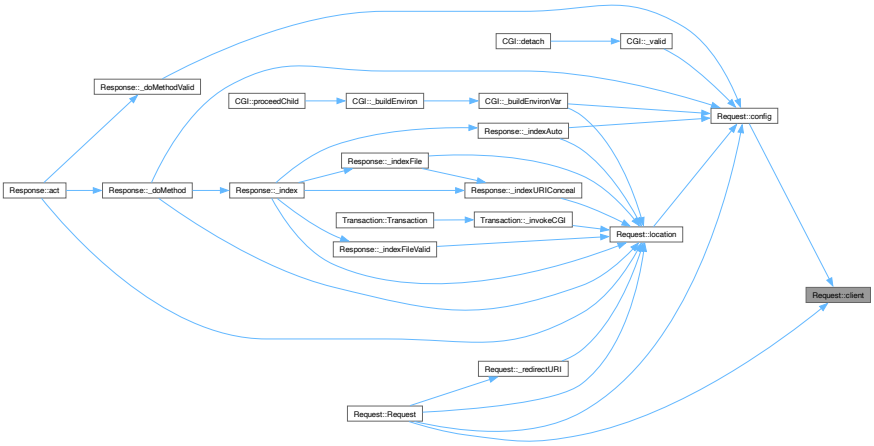
Here is the caller graph for this function:



6.17.2.12 client()

```
const Client & Request::client (
            void ) const
```

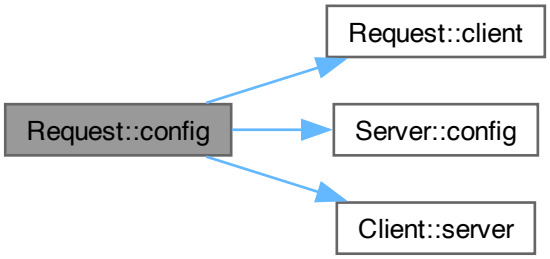
Here is the caller graph for this function:



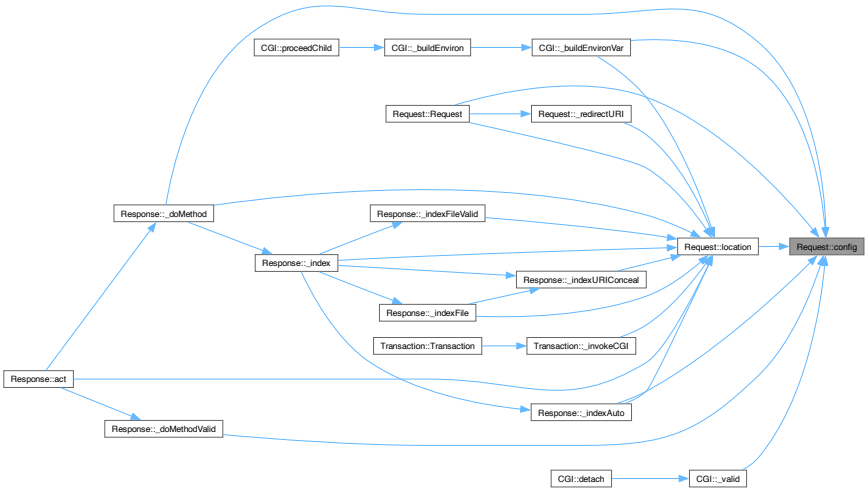
6.17.2.13 config()

```
const config_t & Request::config (
            void ) const
```

Here is the call graph for this function:



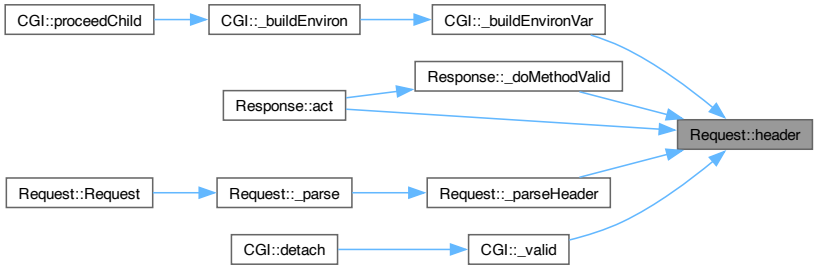
Here is the caller graph for this function:



6.17.2.14 header()

```
const request_header_t & Request::header (
            void ) const
```

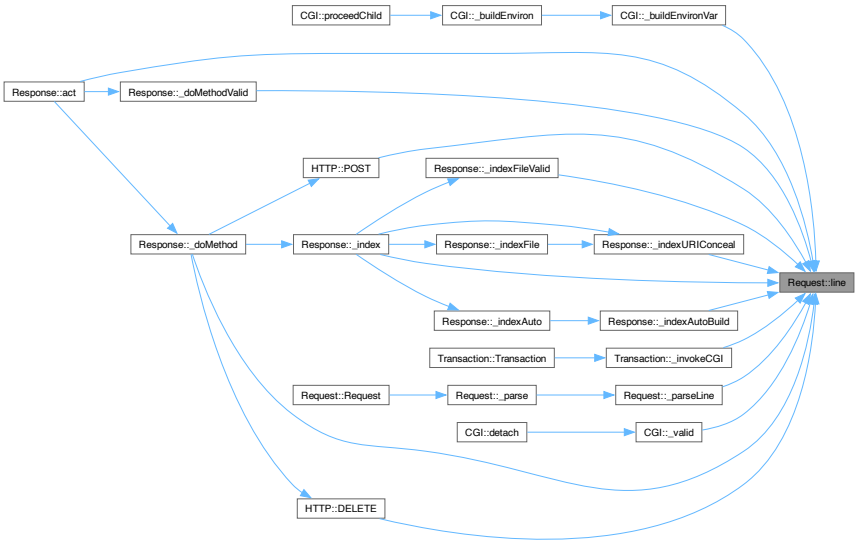
Here is the caller graph for this function:



6.17.2.15 line()

```
const request_line_t & Request::line (
            void ) const
```

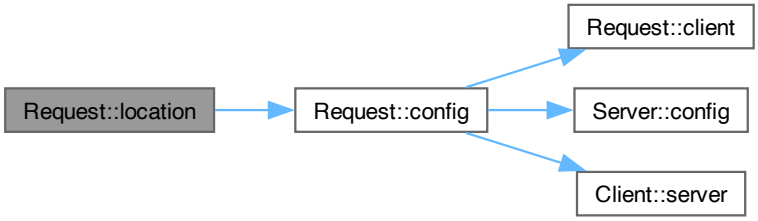
Here is the caller graph for this function:



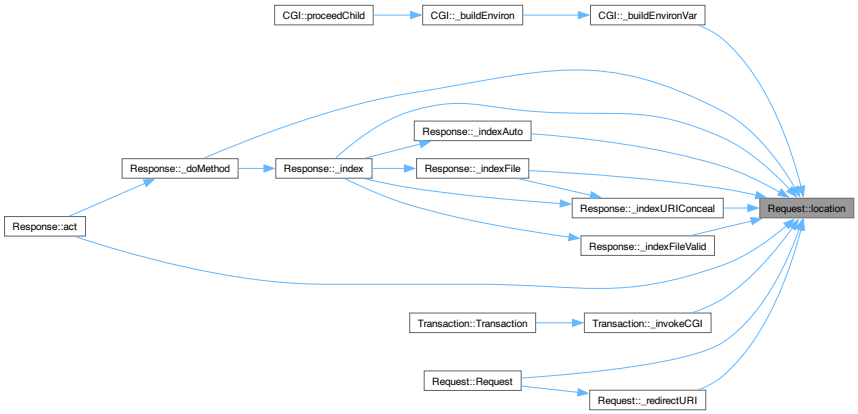
6.17.2.16 location()

```
const location_t & Request::location (
            void ) const
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.17.3 Member Data Documentation

6.17.3.1 _client

```
const Client& Request::_client  [private]
```

6.17.3.2 _config

```
size_t Request::_config  [private]
```

6.17.3.3 _header

```
request_header_t Request::_header  [private]
```

6.17.3.4 _line

```
request_line_t Request::_line  [private]
```

6.17.3.5 _location

```
size_t Request::_location  [private]
```

6.17.3.6 info

```
fstat_t Request::info
```

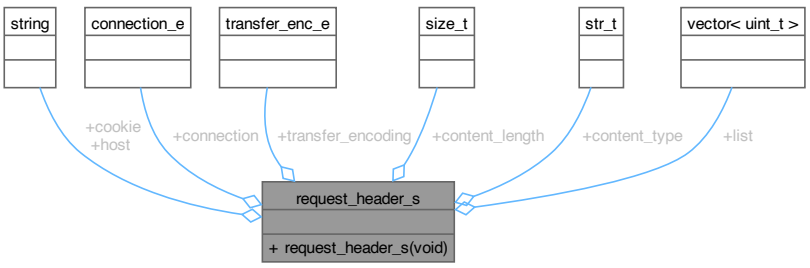
The documentation for this class was generated from the following files:

- src/http/Request.hpp
- src/http/Request.cpp

6.18 request_header_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for request_header_s:



Public Member Functions

- request_header_s (void)

Public Attributes

- str_t host
- connection_e connection
- transfer_enc_e transfer_encoding
- size_t content_length
- type_t content_type
- str_t cookie
- vec_uint_t list

6.18.1 Constructor & Destructor Documentation

6.18.1.1 request_header_s()

```
request_header_s::request_header_s (
            void )
```

6.18.2 Member Data Documentation

6.18.2.1 connection

```
connection_e request_header_s::connection
```

6.18.2.2 content_length

```
size_t request_header_s::content_length
```

6.18.2.3 content_type

```
type_t request_header_s::content_type
```

6.18.2.4 cookie

```
str_t request_header_s::cookie
```

6.18.2.5 host

```
str_t request_header_s::host
```

6.18.2.6 list

```
vec_uint_t request_header_s::list
```

6.18.2.7 transfer_encoding

```
transfer_enc_e request_header_s::transfer_encoding
```

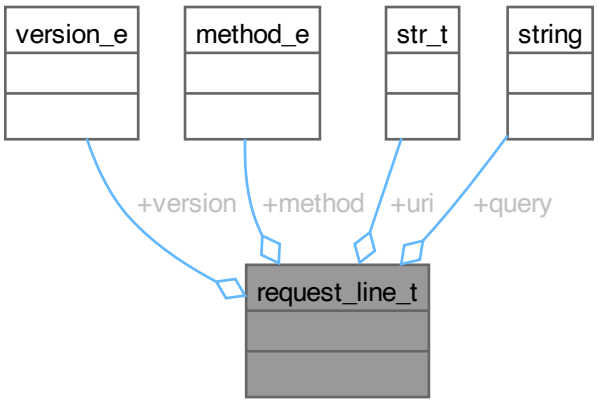
The documentation for this struct was generated from the following files:

- src/http/filter.hpp
- src/http/Request.cpp

6.19 request_line_t Struct Reference

`#include <filter.hpp>`

Collaboration diagram for request_line_t:



Public Attributes

- version_e version
- method_e method
- path_t uri
- str_t query

6.19.1 Member Data Documentation

6.19.1.1 method

`method_e request_line_t::method`

6.19.1.2 query

`str_t request_line_t::query`

6.19.1.3 uri

`path_t request_line_t::uri`

6.19.1.4 version

```
version_e request_line_t::version
```

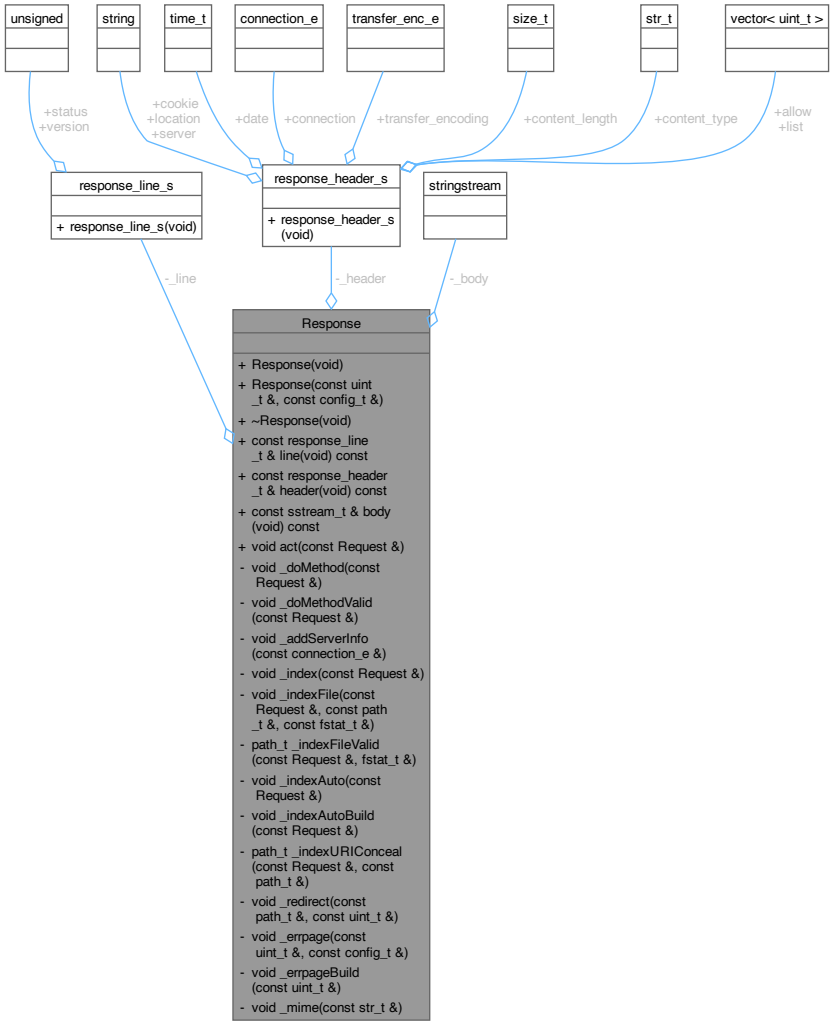
The documentation for this struct was generated from the following file:

- src/http/filter.hpp

6.20 Response Class Reference

```
#include <Response.hpp>
```

Collaboration diagram for Response:



Public Member Functions

- Response (void)
- Response (const uint_t &, const config_t &)
- ~Response (void)
- const response_line_t & line (void) const
- const response_header_t & header (void) const
- const sstream_t & body (void) const
- void act (const Request &)

Private Member Functions

- void _doMethod (const Request &)
- void _doMethodValid (const Request &)
- void _addServerInfo (const connection_e &)
- void _index (const Request &)
- void _indexFile (const Request &, const path_t &, const fstat_t &)
- path_t _indexFileValid (const Request &, fstat_t &)
- void _indexAuto (const Request &)
- void _indexAutoBuild (const Request &)
- path_t _indexURIConceal (const Request &, const path_t &)
- void _redirect (const path_t &, const uint_t &)
- void _errpage (const uint_t &, const config_t &)
- void _errpageBuild (const uint_t &)
- void _mime (const str_t &)

Private Attributes

- response_line_t _line
- response_header_t _header
- sstream_t _body

6.20.1 Constructor & Destructor Documentation

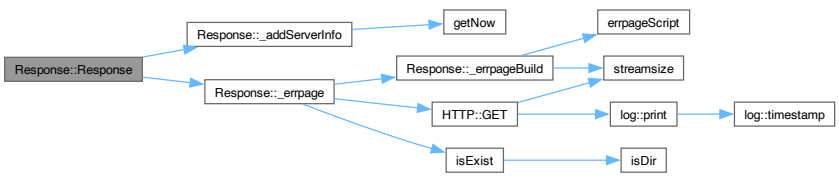
6.20.1.1 Response() [1/2]

```
Response::Response (
            void )
```

6.20.1.2 Response() [2/2]

```
Response::Response (
            const uint_t & status,
            const config_t & conf)
```

Here is the call graph for this function:



6.20.1.3 ~Response()

```
Response::~Response (
            void )
```

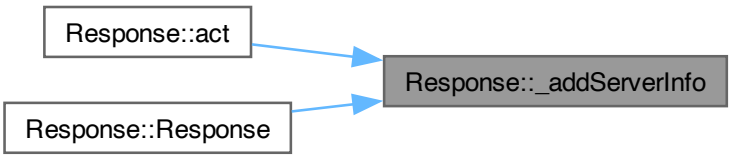
6.20.2 Member Function Documentation

6.20.2.1 _addServerInfo()

```
void Response::_addServerInfo (
            const connection_e & connection)  [private]
```

Here is the call graph for this function:

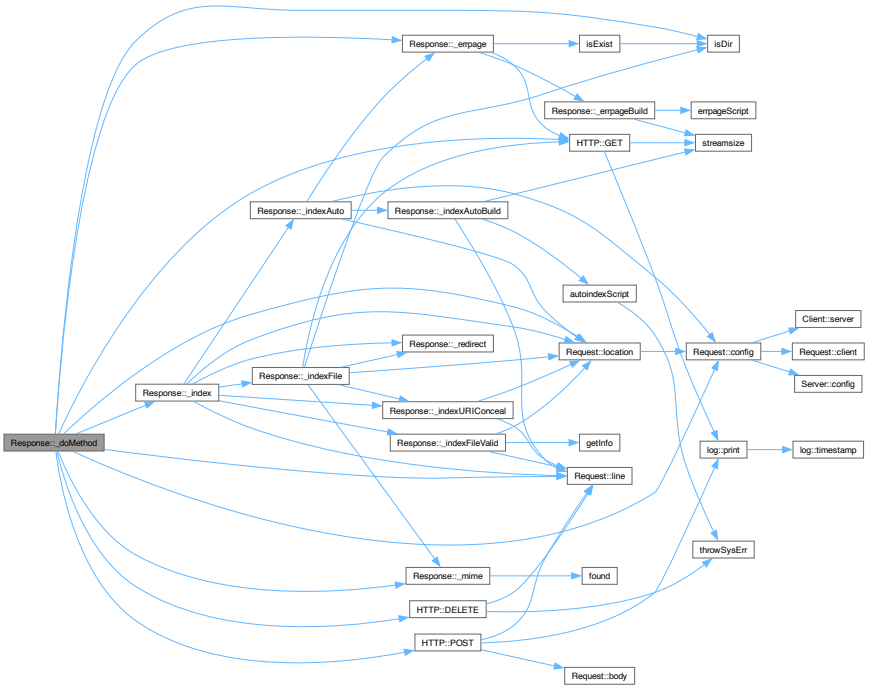
Here is the caller graph for this function:



6.20.2.2 _doMethod()

```
void Response::_doMethod (
            const Request & rqst)  [private]
```

Here is the call graph for this function:

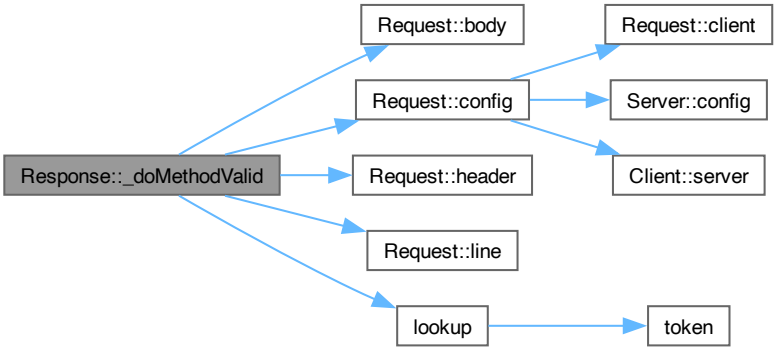


Here is the caller graph for this function:

6.20.2.3 _doMethodValid()

```
void Response::_doMethodValid (
            const Request & rqst)  [private]
```

Here is the call graph for this function:

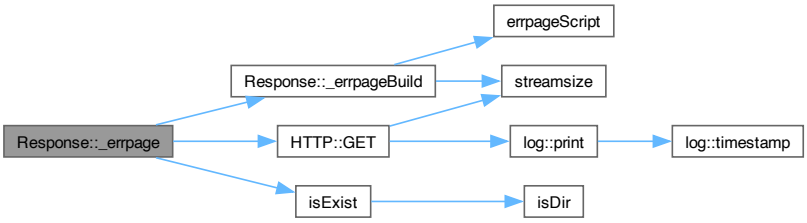


Here is the caller graph for this function:

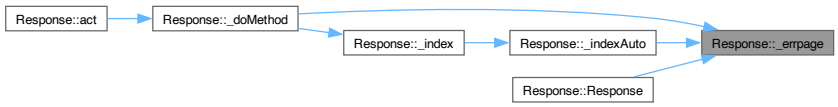
6.20.2.4 _errpage()

```
void Response::_errpage (
            const uint_t & status,
            const config_t & config)  [private]
```

Here is the call graph for this function:



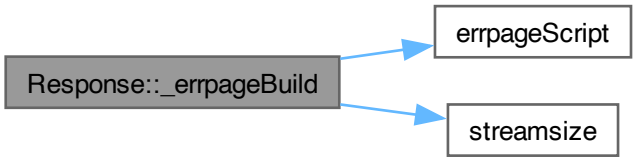
Here is the caller graph for this function:



6.20.2.5 _errpageBuild()

```
void Response::_errpageBuild (
            const uint_t & status)  [private]
```

Here is the call graph for this function:

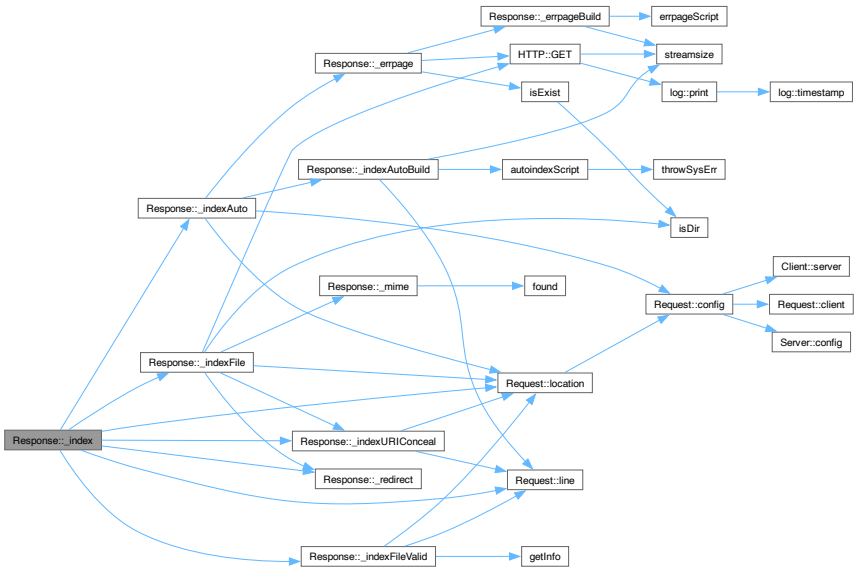


Here is the caller graph for this function:

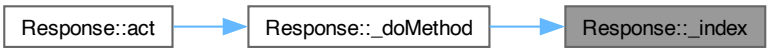
6.20.2.6 _index()

```
void Response::_index (
            const Request & rqst)  [private]
```

Here is the call graph for this function:



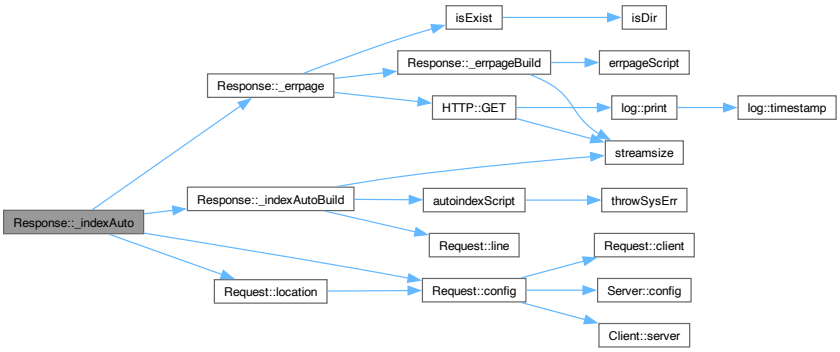
Here is the caller graph for this function:



6.20.2.7 _indexAuto()

```
void Response::_indexAuto (
            const Request & rqst)  [private]
```

Here is the call graph for this function:

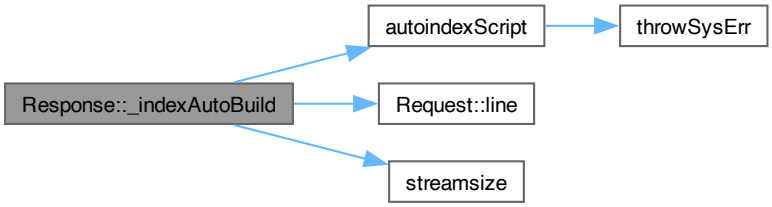


Here is the caller graph for this function:

6.20.2.8 _indexAutoBuild()

```
void Response::_indexAutoBuild (
            const Request & rqst)  [private]
```

Here is the call graph for this function:

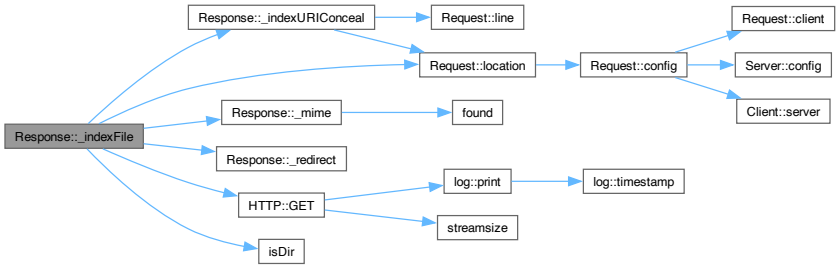


Here is the caller graph for this function:

6.20.2.9 _indexFile()

```
void Response::_indexFile (
            const Request & rqst,
            const path_t & index,
            const fstat_t & info)  [private]
```

Here is the call graph for this function:

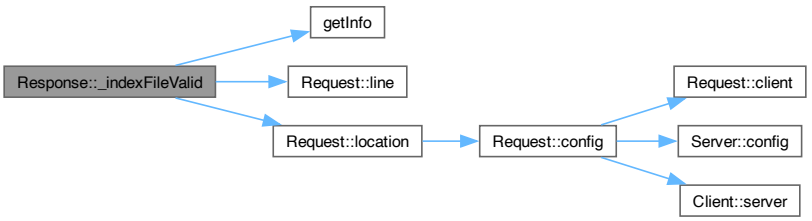


Here is the caller graph for this function:

6.20.2.10 _indexFileValid()

```
path_t Response::_indexFileValid (
            const Request & rqst,
            fstat_t & info)  [private]
```

Here is the call graph for this function:

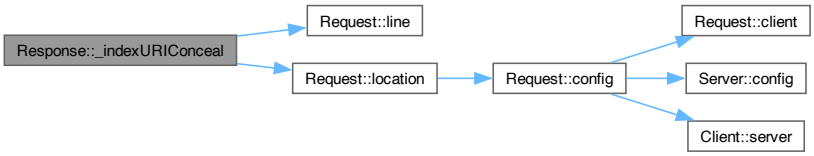


Here is the caller graph for this function:

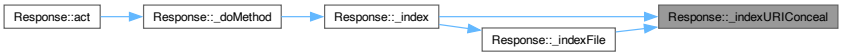
6.20.2.11 _indexURIConceal()

```
path_t Response::_indexURIConceal (
            const Request & rqst,
            const path_t & index)  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

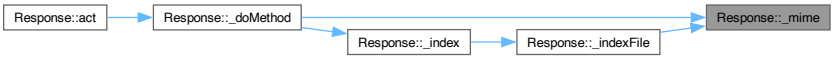


6.20.2.12 _mime()

```
void Response::_mime (
            const str_t & uri)  [private]
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.20.2.13 _redirect()

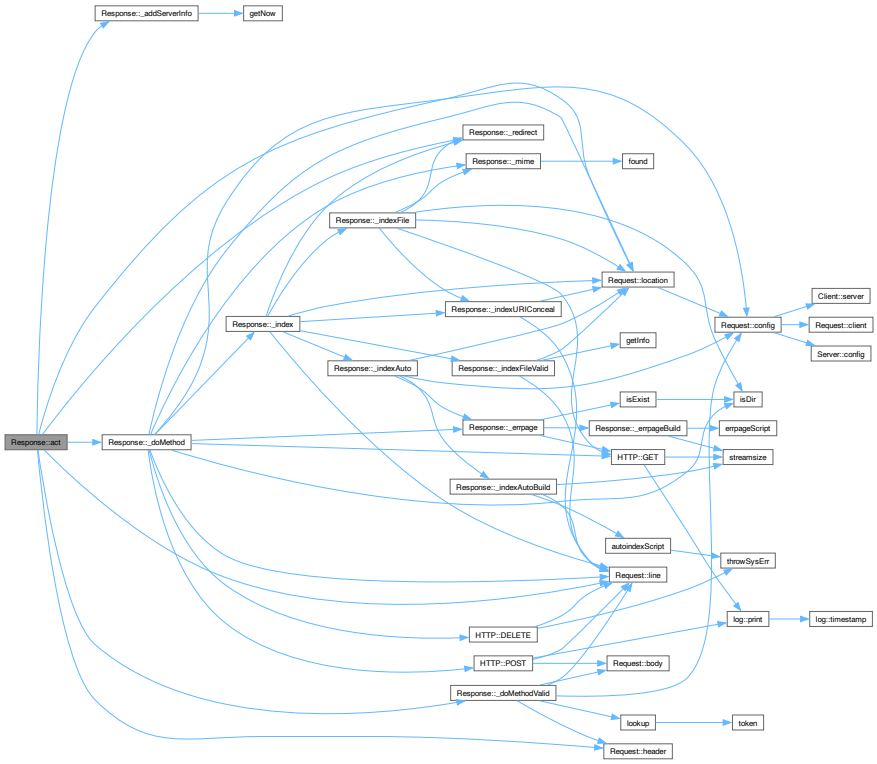
```
void Response::_redirect (
            const path_t & dest,
            const uint_t & status)  [private]
```

Here is the caller graph for this function:

6.20.2.14 act()

```
void Response::act (
            const Request & rqst)
```

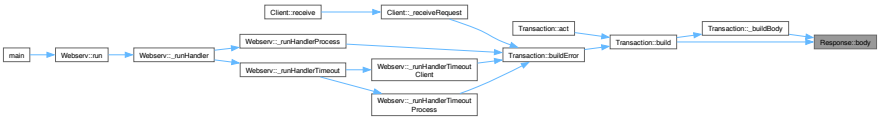
Here is the call graph for this function:



6.20.2.15 body()

```
const sstream_t & Response::body (
            void ) const
```

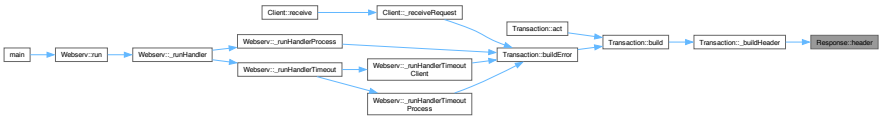
Here is the caller graph for this function:



6.20.2.16 header()

```
const response_header_t & Response::header (
            void ) const
```

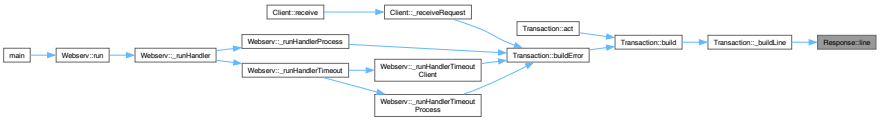
Here is the caller graph for this function:



6.20.2.17 line()

```
const response_line_t & Response::line (
            void ) const
```

Here is the caller graph for this function:



6.20.3 Member Data Documentation

6.20.3.1 _body

```
sstream_t Response::_body  [private]
```

6.20.3.2 _header

```
response_header_t Response::_header  [private]
```

6.20.3.3 _line

```
response_line_t Response::_line  [private]
```

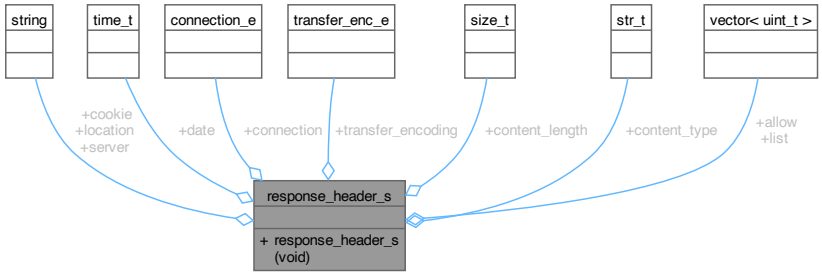
The documentation for this class was generated from the following files:

- src/http/Response.hpp
- src/http/Response.cpp

6.21 response_header_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for response_header_s:



Public Member Functions

- response_header_s (void)

Public Attributes

- str_t server
- ctime_t date
- connection_e connection
- transfer_enc_e transfer_encoding
- size_t content_length
- type_t content_type
- str_t location
- vec_uint_t allow
- str_t cookie
- vec_uint_t list

6.21.1 Constructor & Destructor Documentation

6.21.1.1 response_header_s()

```
response_header_s::response_header_s (
            void )
```

6.21.2 Member Data Documentation

6.21.2.1 allow

```
vec_uint_t response_header_s::allow
```

6.21.2.2 connection

```
connection_e response_header_s::connection
```

6.21.2.3 content_length

```
size_t response_header_s::content_length
```

6.21.2.4 content_type

```
type_t response_header_s::content_type
```

6.21.2.5 cookie

```
str_t response_header_s::cookie
```

6.21.2.6 date

```
ctime_t response_header_s::date
```

6.21.2.7 list

```
vec_uint_t response_header_s::list
```

6.21.2.8 location

```
str_t response_header_s::location
```

6.21.2.9 server

```
str_t response_header_s::server
```

6.21.2.10 transfer_encoding

```
transfer_enc_e response_header_s::transfer_encoding
```

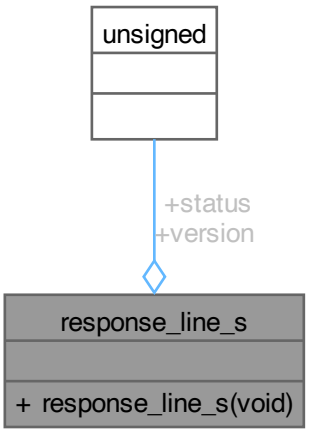
The documentation for this struct was generated from the following files:

- src/http/filter.hpp
- src/http/Response.cpp

6.22 response_line_s Struct Reference

```
#include <filter.hpp>
```

Collaboration diagram for response_line_s:



Public Member Functions

- response_line_s (void)

Public Attributes

- unsigned version: 2
- uint_t status

6.22.1 Constructor & Destructor Documentation

6.22.1.1 response_line_s()

```
response_line_s::response_line_s (
            void )
```

6.22.2 Member Data Documentation

6.22.2.1 status

```
uint_t response_line_s::status
```

6.22.2.2 version

```
unsigned response_line_s::version
```

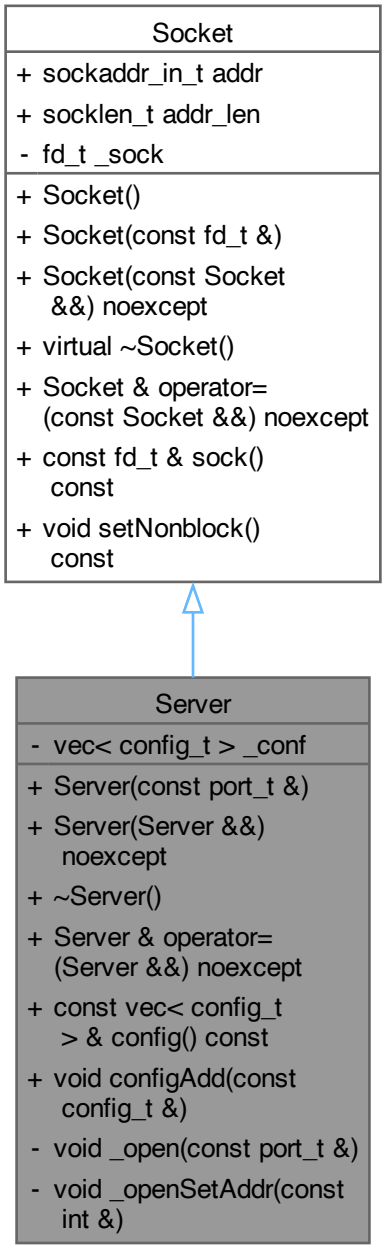
The documentation for this struct was generated from the following files:

- src/http/filter.hpp
- src/http/Response.cpp

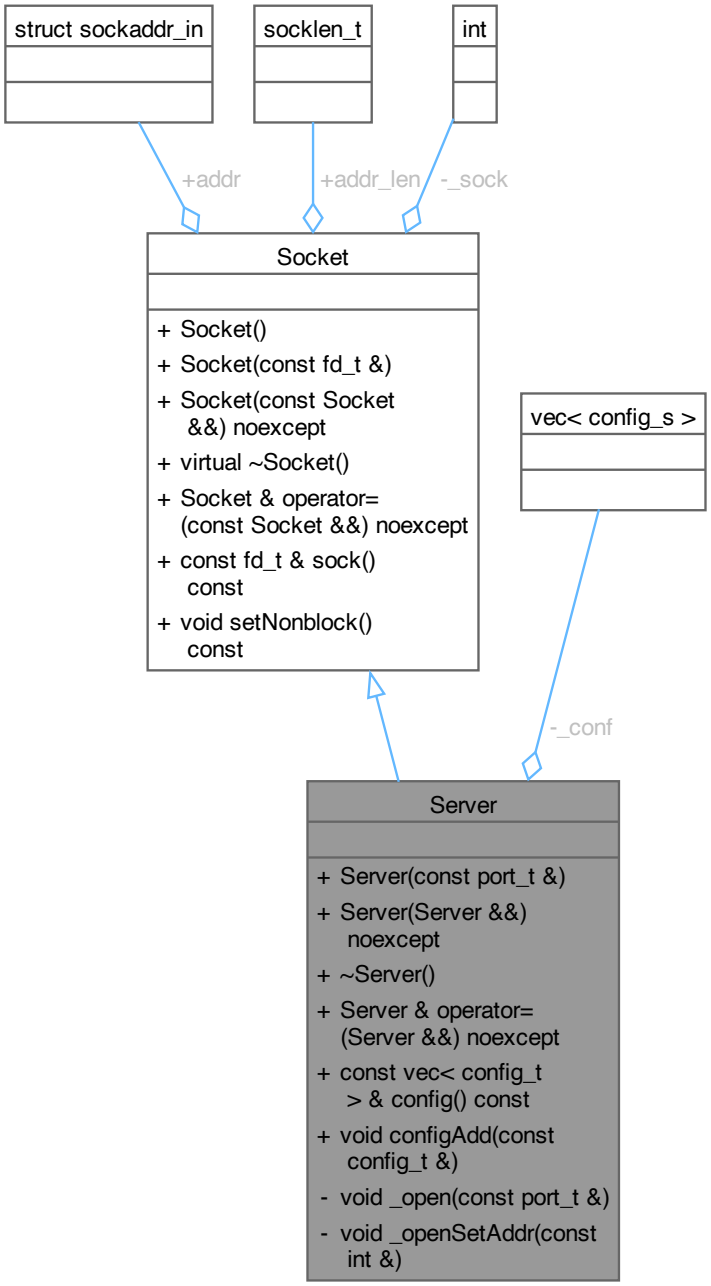
6.23 Server Class Reference

```
#include <Server.hpp>
```

Inheritance diagram for Server:



Collaboration diagram for Server:



Public Member Functions

- Server (const port_t &)
- Server (Server &&) noexcept
- ~Server ()
- Server & operator= (Server &&) noexcept
- const vec< config_t > & config () const
- void configAdd (const config_t &)

Public Member Functions inherited from Socket

- Socket ()
- Socket (const fd_t &)
- Socket (const Socket &&) noexcept
- virtual ~Socket ()
- Socket & operator= (const Socket &&) noexcept
- const fd_t & sock () const
- void setNonblock () const

Private Member Functions

- void _open (const port_t &)
- void _openSetAddr (const int &)

Private Attributes

- vec< config_t > _conf

Additional Inherited Members

Public Attributes inherited from Socket

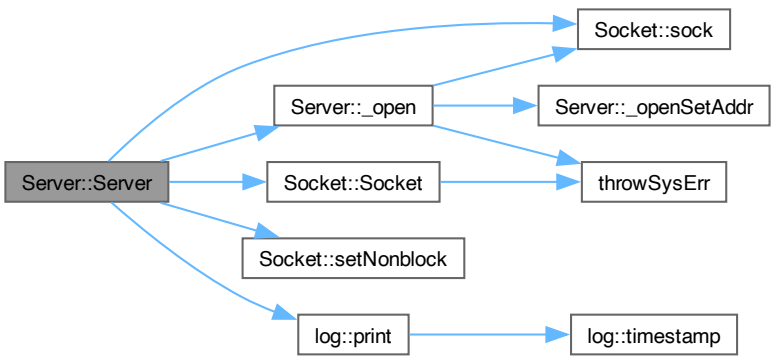
- sockaddr_in_t addr
- socklen_t addr_len

6.23.1 Constructor & Destructor Documentation

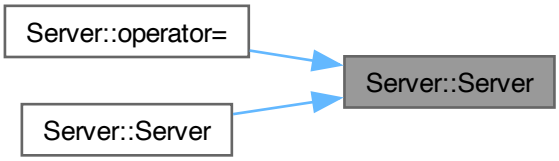
6.23.1.1 Server() [1/2]

```
Server::Server (
            const port_t & port)
```

Here is the call graph for this function:



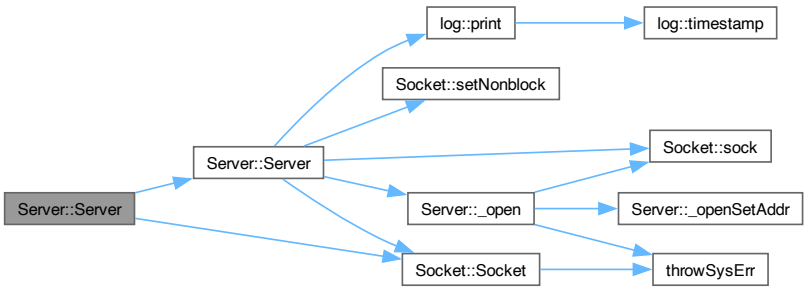
Here is the caller graph for this function:



6.23.1.2 Server() [2/2]

```
Server::Server (
            Server && target)  [noexcept]
```

Here is the call graph for this function:



6.23.1.3 ~Server()

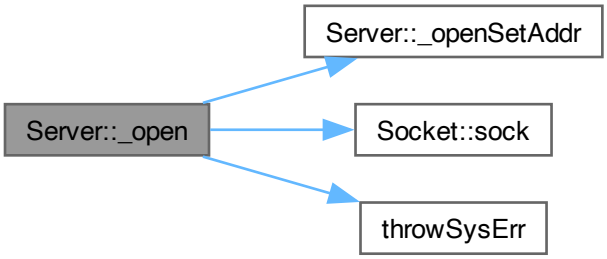
```
Server::~Server ()
```

6.23.2 Member Function Documentation

6.23.2.1 _open()

```
void Server::_open (
            const port_t & port)  [private]
```

Here is the call graph for this function:

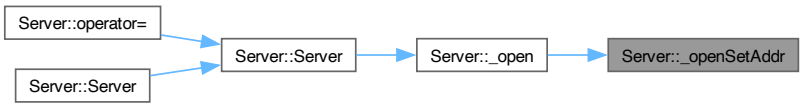


Here is the caller graph for this function:

6.23.2.2 _openSetAddr()

```
void Server::_openSetAddr (
            const int & port)  [private]
```

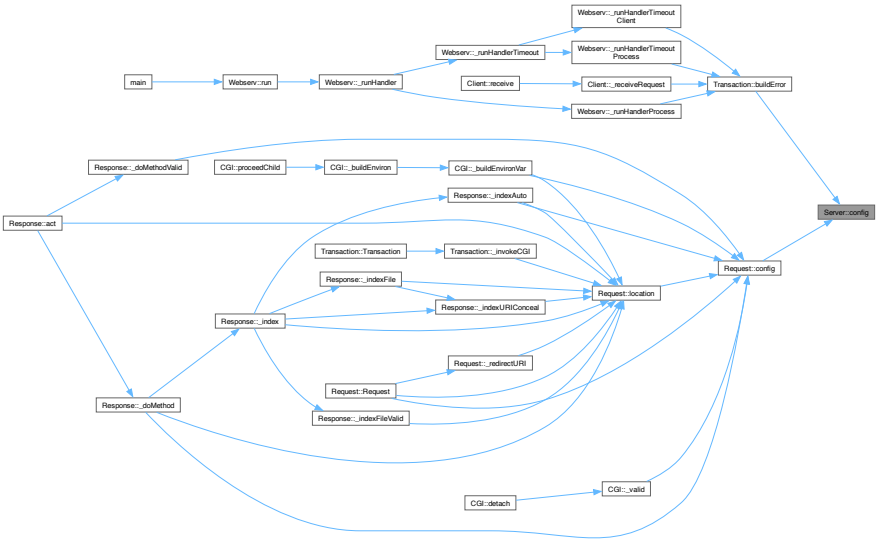
Here is the caller graph for this function:



6.23.2.3 config()

```
const vec< config_t > & Server::config () const
```

Here is the caller graph for this function:



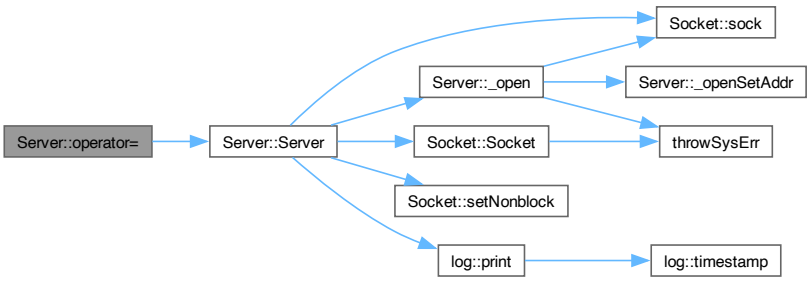
6.23.2.4 configAdd()

```
void Server::configAdd (
            const config_t & conf)
```

6.23.2.5 operator=()

```
Server & Server::operator= (
            Server && target)  [noexcept]
```

Here is the call graph for this function:



6.23.3 Member Data Documentation

6.23.3.1 _conf

```
vec<config_t> Server::_conf  [private]
```

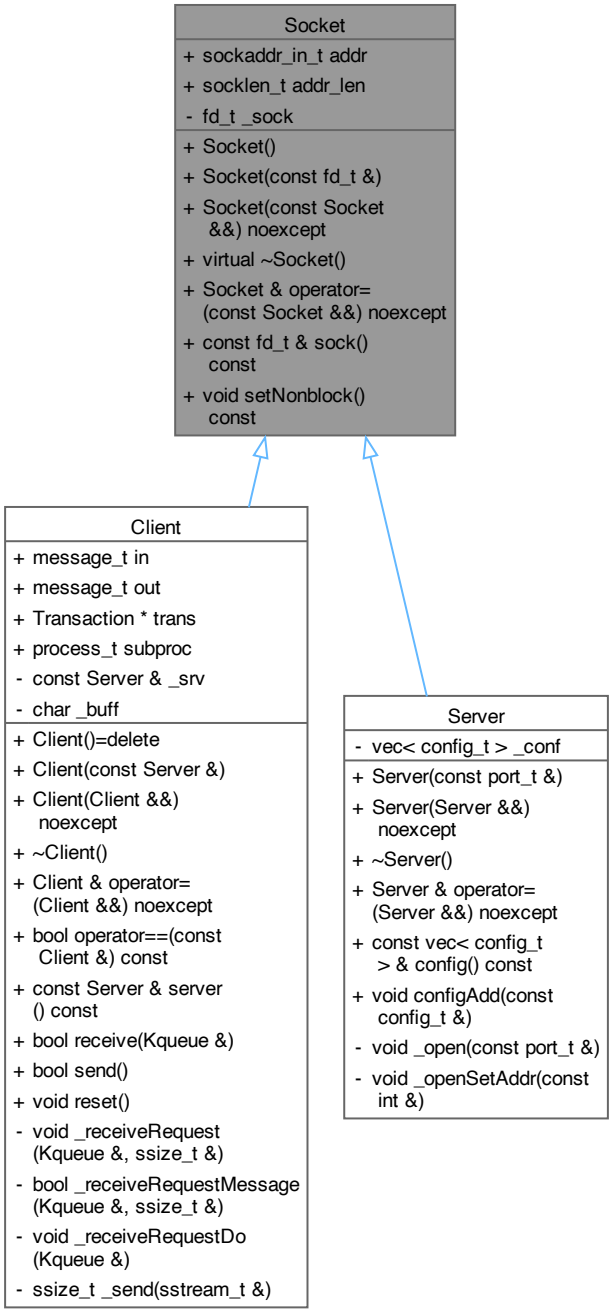
The documentation for this class was generated from the following files:

- src/core/Server.hpp
- src/core/Server.cpp

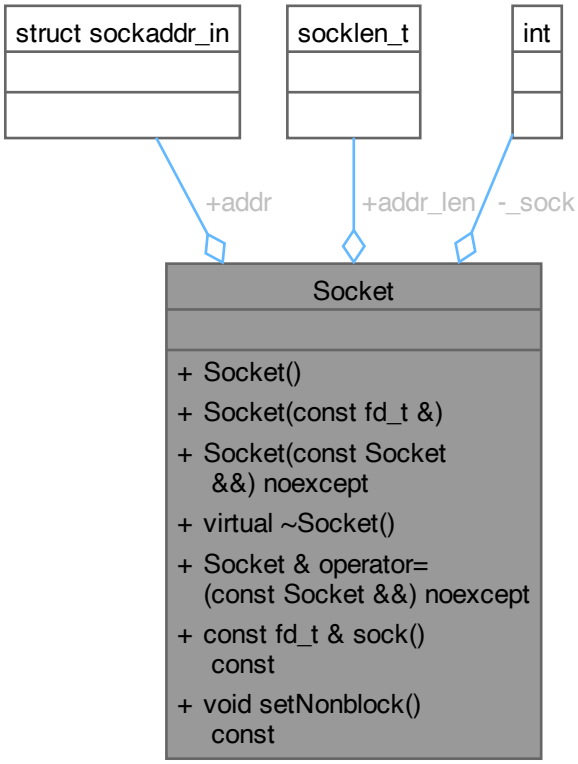
6.24 Socket Class Reference

```
#include <Socket.hpp>
```

Inheritance diagram for Socket:



Collaboration diagram for Socket:



Public Member Functions

- Socket ()
- Socket (const fd_t &)
- Socket (const Socket &&) noexcept
- virtual ~Socket ()
- Socket & operator= (const Socket &&) noexcept
- const fd_t & sock () const
- void setNonblock () const

Public Attributes

- sockaddr_in_t addr
- socklen_t addr_len

Private Attributes

- fd_t _sock

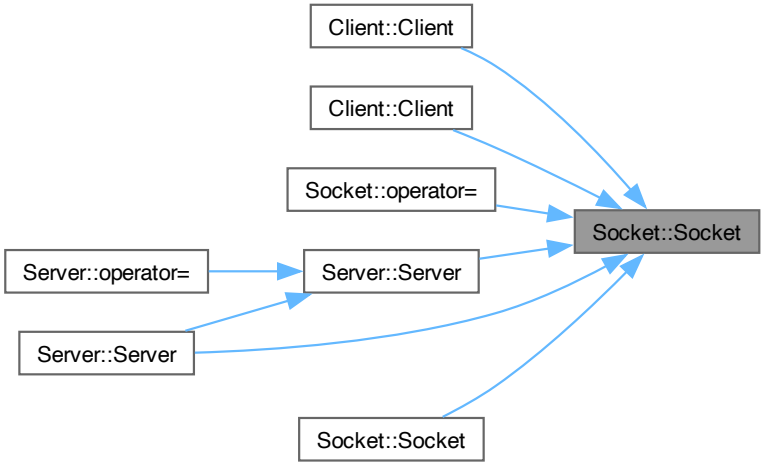
6.24.1 Constructor & Destructor Documentation

6.24.1.1 Socket() [1/3]

```
Socket::Socket ()
```

Here is the call graph for this function:

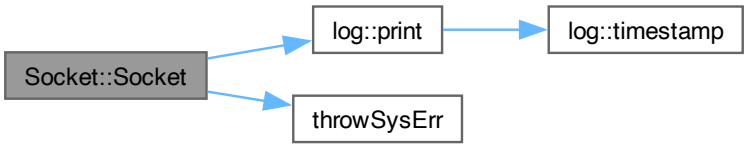
Here is the caller graph for this function:



6.24.1.2 Socket() [2/3]

```
Socket::Socket (
            const fd_t & sock_srv)
```

Here is the call graph for this function:



6.24.1.3 Socket() [3/3]

```
Socket::Socket (
            const Socket && source)  [noexcept]
```

Here is the call graph for this function:

6.24.1.4 ~Socket()

```
Socket::~Socket ()  [virtual]
```

6.24.2 Member Function Documentation

6.24.2.1 operator=()

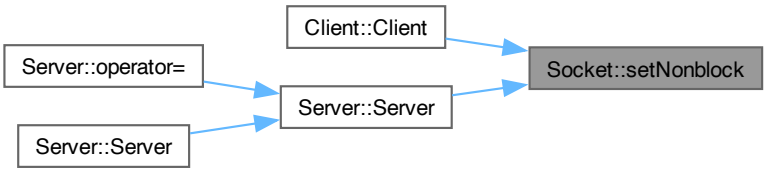
```
Socket & Socket::operator= (
            const Socket && source)  [noexcept]
```

Here is the call graph for this function:

6.24.2.2 setNonblock()

```
void Socket::setNonblock () const
```

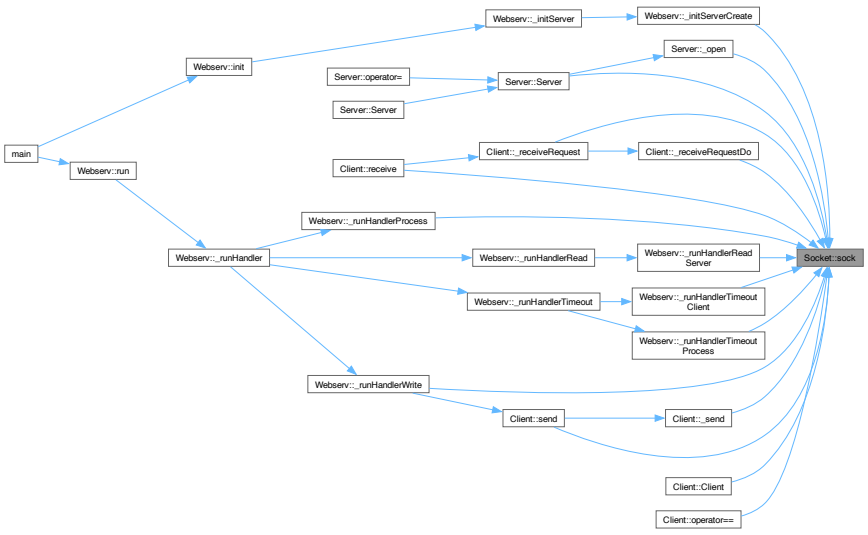
Here is the caller graph for this function:



6.24.2.3 sock()

```
const fd_t & Socket::sock () const
```

Here is the caller graph for this function:



6.24.3 Member Data Documentation

6.24.3.1 _sock

```
fd_t Socket::_sock  [private]
```

6.24.3.2 addr

```
sockaddr_in_t Socket::addr
```

6.24.3.3 addr_len

```
socklen_t Socket::addr_len
```

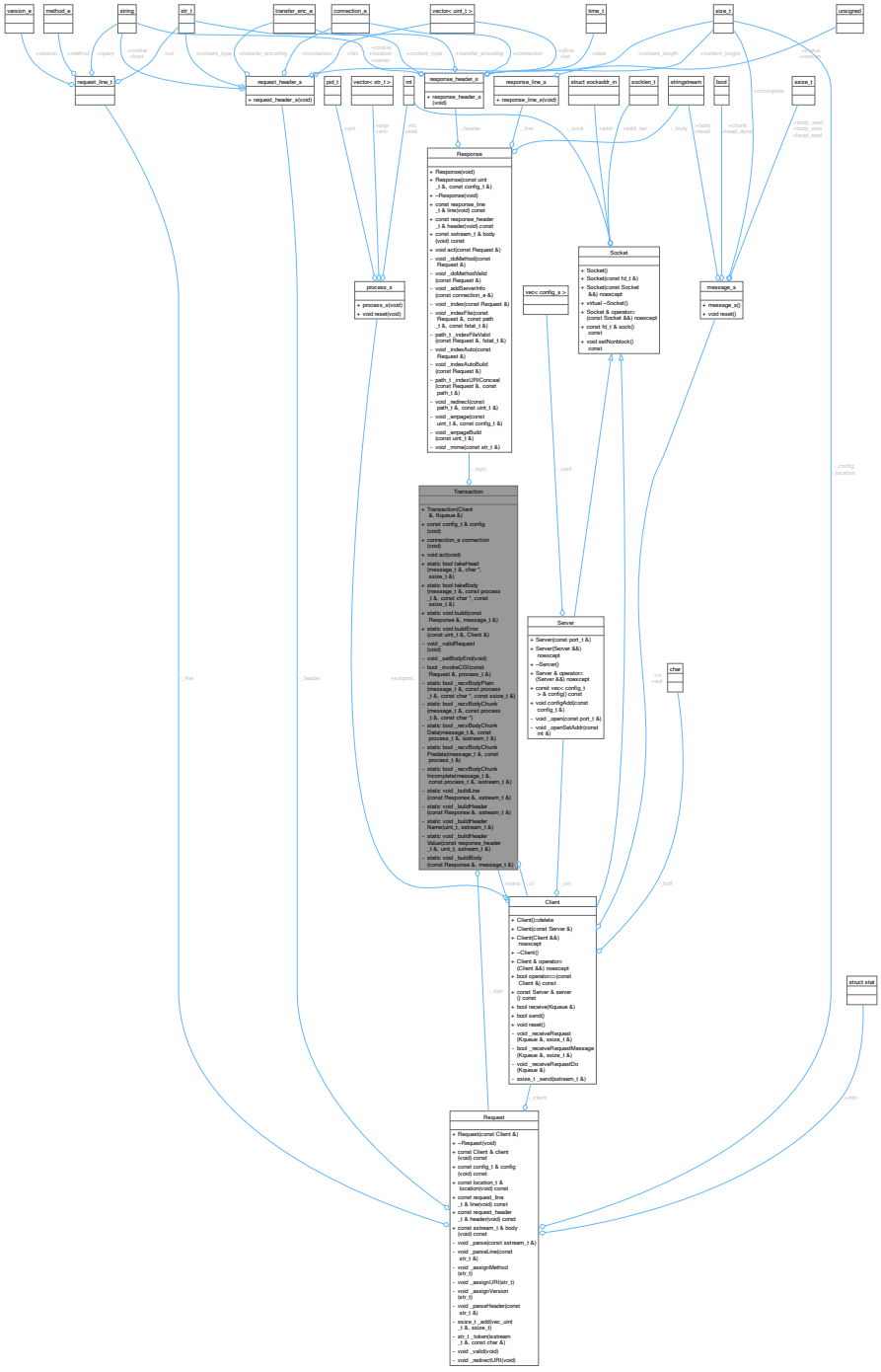
The documentation for this class was generated from the following files:

- src/common/Socket.hpp
- src/common/Socket.cpp

6.25 Transaction Class Reference

```
#include <Transaction.hpp>
```

Collaboration diagram for Transaction:



Public Member Functions

- Transaction (Client &, Kqueue &)
- const config_t & config (void)
- connection_e connection (void)
- void act (void)

Static Public Member Functions

- static bool takeHead (message_t &, char *, ssize_t &)
- static bool takeBody (message_t &, const process_t &, const char *, const ssize_t &)
- static void build (const Response &, message_t &)
- static void buildError (const uint_t &, Client &)

Private Member Functions

- void _validRequest (void)
- void _setBodyEnd (void)
- bool _invokeCGI (const Request &, process_t &)

Static Private Member Functions

- static bool _recvBodyPlain (message_t &, const process_t &, const char *, const ssize_t &)
- static bool _recvBodyChunk (message_t &, const process_t &, const char *)
- static bool _recvBodyChunkData (message_t &, const process_t &, isstream_t &)
- static bool _recvBodyChunkPredata (message_t &, const process_t &)
- static bool _recvBodyChunkIncomplete (message_t &, const process_t &, isstream_t &)
- static void _buildLine (const Response &, sstream_t &)
- static void _buildHeader (const Response &, sstream_t &)
- static void _buildHeaderName (uint_t, sstream_t &)
- static void _buildHeaderValue (const response_header_t &, uint_t, sstream_t &)
- static void _buildBody (const Response &, message_t &)

Private Attributes

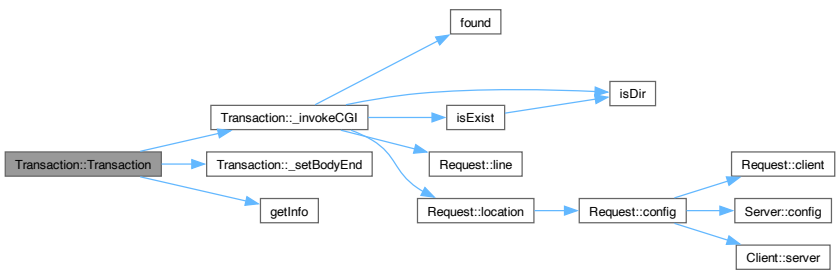
- Client & _cl
- Request _rqst
- Response _rspn

6.25.1 Constructor & Destructor Documentation

6.25.1.1 Transaction()

```
Transaction::Transaction (
            Client & client,
            Kqueue & evnt)
```

Here is the call graph for this function:



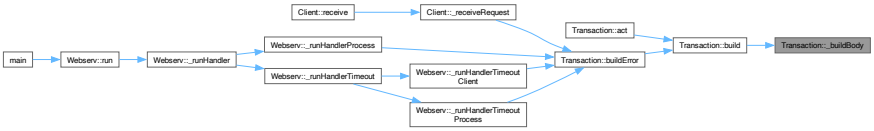
6.25.2 Member Function Documentation

6.25.2.1 _buildBody()

```
void Transaction::_buildBody (
            const Response & rspn,
            message_t & out)  [static], [private]
```

Here is the call graph for this function:

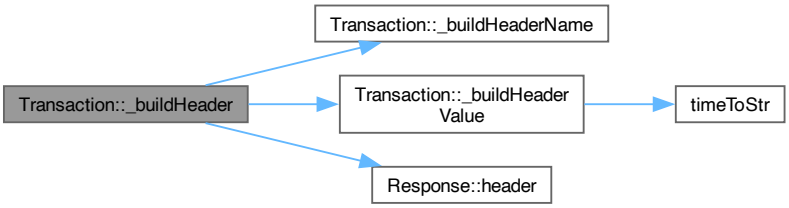
Here is the caller graph for this function:



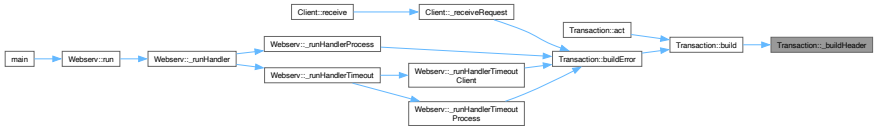
6.25.2.2 _buildHeader()

```
void Transaction::_buildHeader (
            const Response & rspn,
            sstream_t & out_msg)  [static], [private]
```

Here is the call graph for this function:



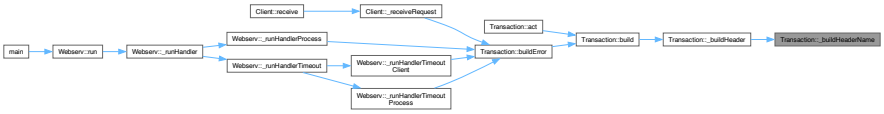
Here is the caller graph for this function:



6.25.2.3 _buildHeaderName()

```
void Transaction::_buildHeaderName (
            uint_t id,
            sstream_t & out_msg)  [static], [private]
```

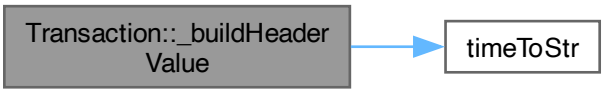
Here is the caller graph for this function:



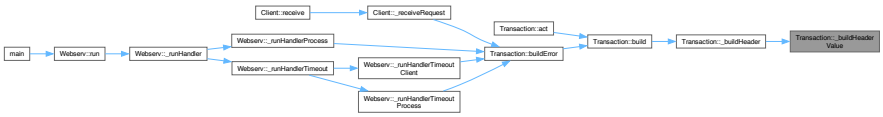
6.25.2.4 _buildHeaderValue()

```
void Transaction::_buildHeaderValue (
            const response_header_t & header,
            uint_t id,
            sstream_t & out_msg)  [static], [private]
```

Here is the call graph for this function:



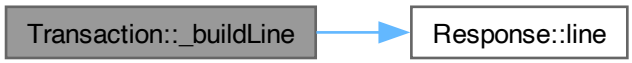
Here is the caller graph for this function:



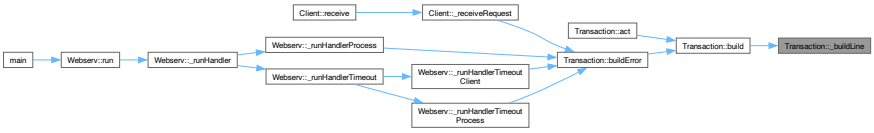
6.25.2.5 _buildLine()

```
void Transaction::_buildLine (
            const Response & rspn,
            sstream_t & out_msg)  [static], [private]
```

Here is the call graph for this function:



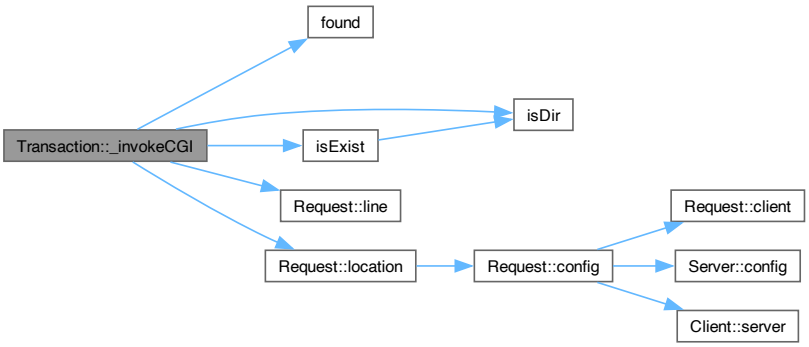
Here is the caller graph for this function:



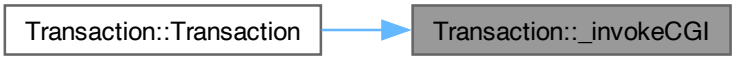
6.25.2.6 _invokeCGI()

```
bool Transaction::_invokeCGI (
            const Request & rqst,
            process_t & procs)  [private]
```

Here is the call graph for this function:



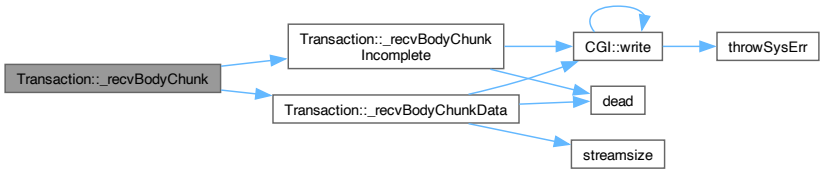
Here is the caller graph for this function:



6.25.2.7 _recvBodyChunk()

```
bool Transaction::_recvBodyChunk (
            message_t & in,
            const process_t & procs,
            const char * buf)  [static], [private]
```

Here is the call graph for this function:

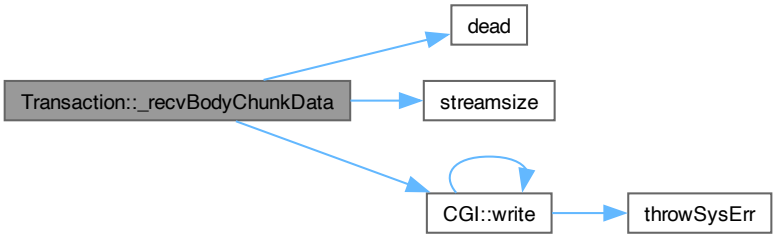


Here is the caller graph for this function:

6.25.2.8 _recvBodyChunkData()

```
bool Transaction::_recvBodyChunkData (
            message_t & in,
            const process_t & procs,
            isstream_t & iss)  [static], [private]
```

Here is the call graph for this function:

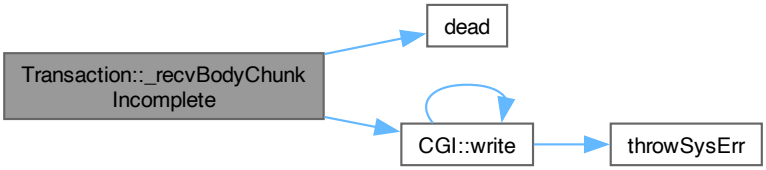


Here is the caller graph for this function:

6.25.2.9 _recvBodyChunkIncomplete()

```
bool Transaction::_recvBodyChunkIncomplete (
            message_t & in,
            const process_t & procs,
            isstream_t & iss)  [static], [private]
```

Here is the call graph for this function:

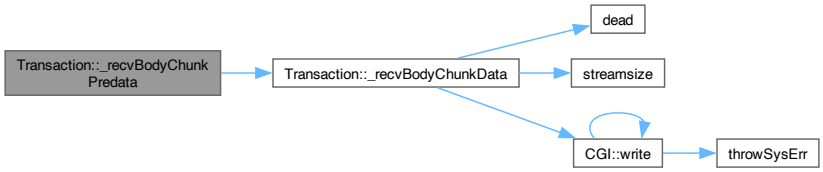


Here is the caller graph for this function:

6.25.2.10 _recvBodyChunkPredata()

```
bool Transaction::_recvBodyChunkPredata (
            message_t & in,
            const process_t & procs)  [static], [private]
```

Here is the call graph for this function:

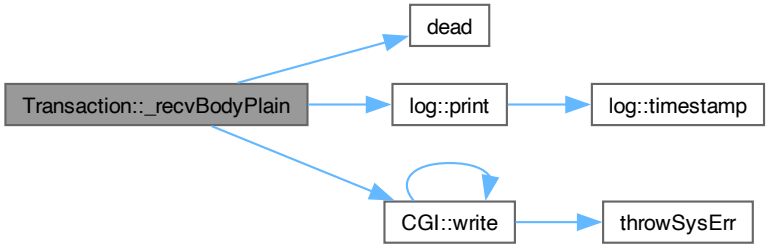


Here is the caller graph for this function:

6.25.2.11 _recvBodyPlain()

```
bool Transaction::_recvBodyPlain (
            message_t & in,
            const process_t & procs,
            const char * buf,
            const ssize_t & byte_read)  [static], [private]
```

Here is the call graph for this function:

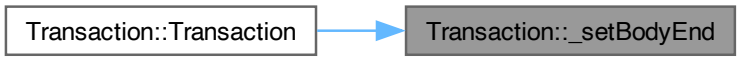


Here is the caller graph for this function:

6.25.2.12 _setBodyEnd()

```
void Transaction::_setBodyEnd (
            void )  [private]
```

Here is the caller graph for this function:



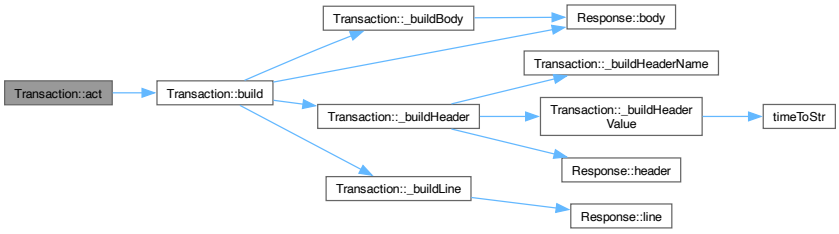
6.25.2.13 _validRequest()

```
void Transaction::_validRequest (
            void )  [private]
```

6.25.2.14 act()

```
void Transaction::act (
            void )
```

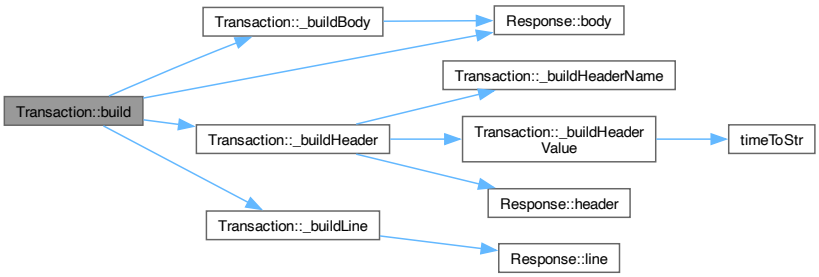
Here is the call graph for this function:



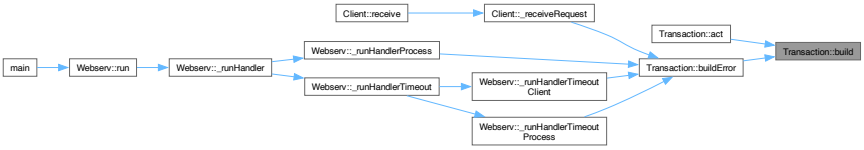
6.25.2.15 build()

```
void Transaction::build (
            const Response & rspn,
            message_t & out)  [static]
```

Here is the call graph for this function:



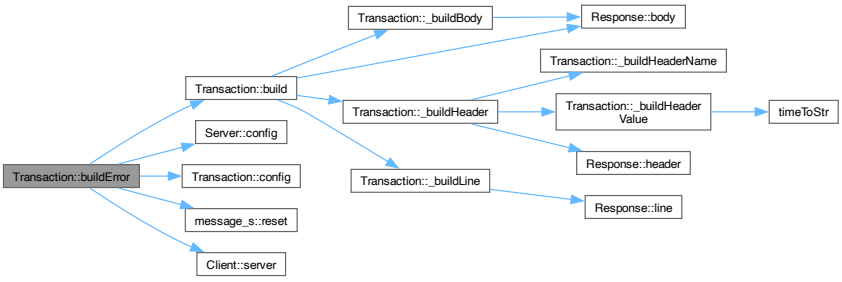
Here is the caller graph for this function:



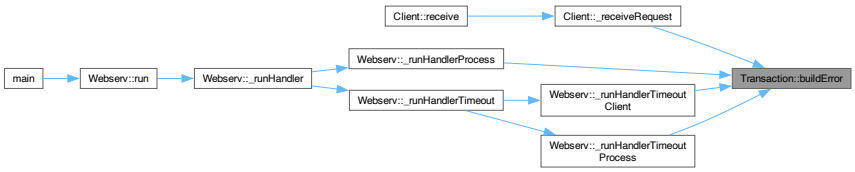
6.25.2.16 buildError()

```
void Transaction::buildError (
            const uint_t & status,
            Client & cl)  [static]
```

Here is the call graph for this function:



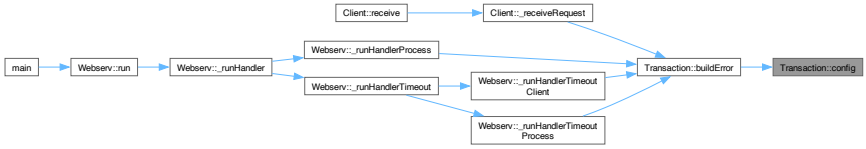
Here is the caller graph for this function:



6.25.2.17 config()

```
const config_t & Transaction::config (
            void )
```

Here is the caller graph for this function:



6.25.2.18 connection()

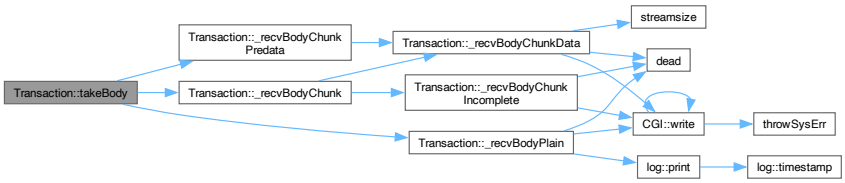
```
connection_e Transaction::connection (
            void )
```

Here is the caller graph for this function:

6.25.2.19 takeBody()

```
bool Transaction::takeBody (
            message_t & in,
            const process_t & procs,
            const char * buf,
            const ssize_t & byte_read)  [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.25.2.20 takeHead()

```
bool Transaction::takeHead (
            message_t & in,
            char * buf,
            ssize_t & byte_read)  [static]
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.25.3 Member Data Documentation

6.25.3.1 _cl

```
Client& Transaction::_cl  [private]
```

6.25.3.2 _rqst

```
Request Transaction::_rqst  [private]
```

6.25.3.3 _rspn

```
Response Transaction::_rspn  [private]
```

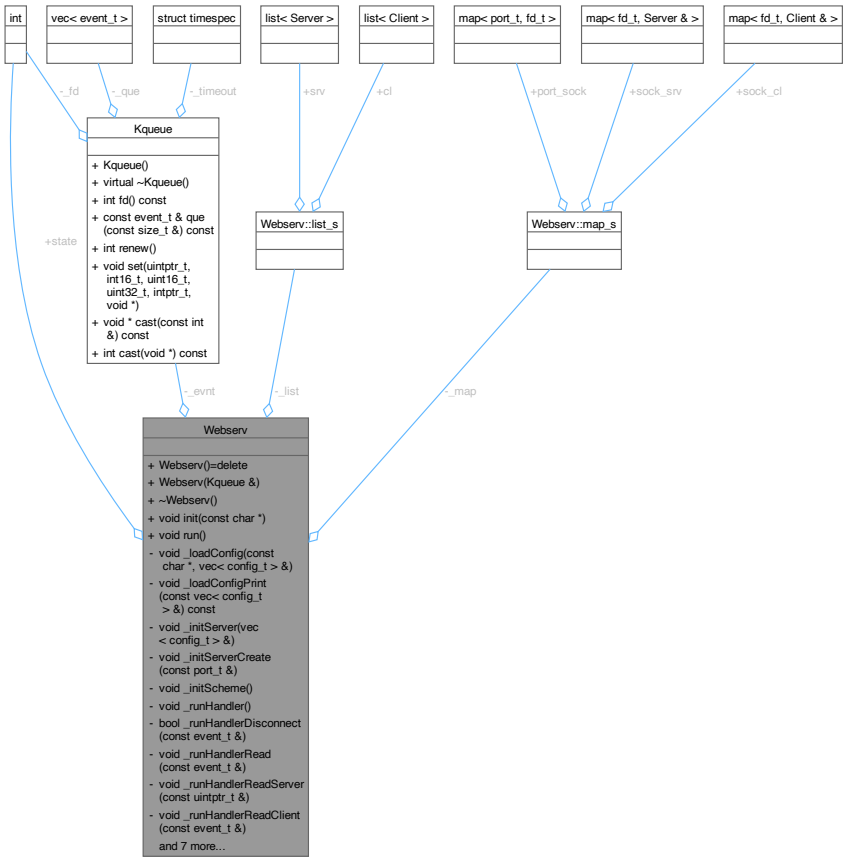
The documentation for this class was generated from the following files:

- src/http/Transaction.hpp
- src/http/Transaction.cpp

6.26 Webserv Class Reference

```
#include <Webserv.hpp>
```

Collaboration diagram for Webserv:



Classes

- struct list_s
- struct map_s

Public Member Functions

- Webserv ()=delete
- Webserv (Kqueue &)

  Construct a new Webserv:: Webserv object.
- ~Webserv ()

  Destroy the Webserv:: Webserv object.
- void init (const char *)
- void run ()

Public Attributes

- int state

Private Member Functions

- void _loadConfig (const char *, vec< config_t > &)
- void _loadConfigPrint (const vec< config_t > &) const
- void _initServer (vec< config_t > &)
- void _initServerCreate (const port_t &)
- void _initScheme ()
- void _runHandler ()
- bool _runHandlerDisconnect (const event_t &)
- void _runHandlerRead (const event_t &)
- void _runHandlerReadServer (const uintptr_t &)
- void _runHandlerReadClient (const event_t &)
- void _runHandlerWrite (const event_t &)
- void _runHandlerProcess (const event_t &)
- void _runHandlerTimeout (const event_t &)
- void _runHandlerTimeoutClient (const event_t &, Client &)
- void _runHandlerTimeoutProcess (const event_t &)
- void _disconnect (const event_t &)
- void _disconnectPrint (const event_t &)

Private Attributes

- Kqueue & _evnt
- struct Webserv::list_s _list
- struct Webserv::map_s _map

6.26.1 Constructor & Destructor Documentation

6.26.1.1 Webserv() [1/2]

```
Webserv::Webserv ()  [delete]
```

6.26.1.2 Webserv() [2/2]

```
Webserv::Webserv (
            Kqueue & event_interface)
```

Construct a new Webserv:: Webserv object.

Parameters

event_interface	

6.26.1.3 ~Webserv()

```
Webserv::~Webserv ()
```

Destroy the Webserv:: Webserv object.

6.26.2 Member Function Documentation

6.26.2.1 _disconnect()

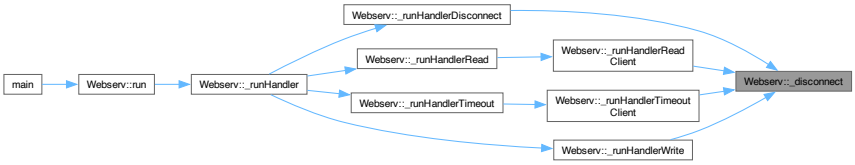
```
void Webserv::_disconnect (
            const event_t & ev)  [private]
```

Parameters

ev	

Here is the call graph for this function:

Here is the caller graph for this function:



6.26.2.2 _disconnectPrint()

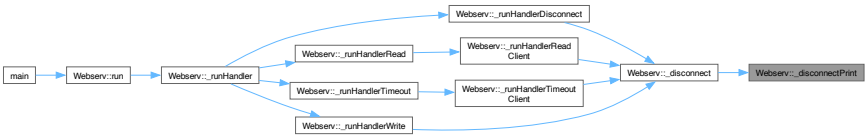
```
void Webserv::_disconnectPrint (
            const event_t & ev)  [private]
```

Parameters

ev	

Here is the call graph for this function:

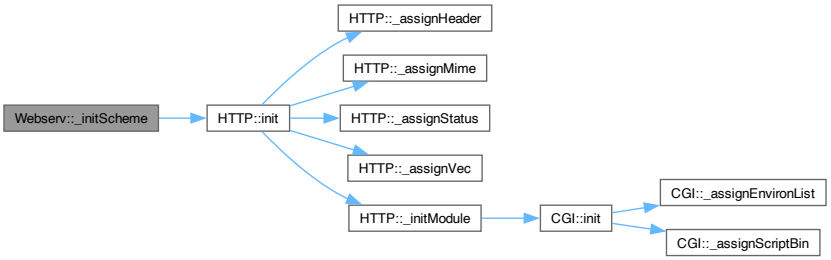
Here is the caller graph for this function:



6.26.2.3 _initScheme()

```
void Webserv::_initScheme () [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:

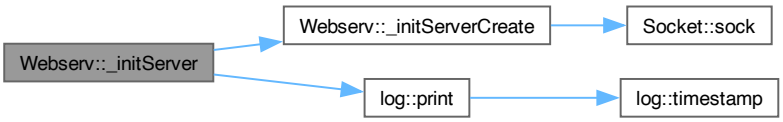
6.26.2.4 _initServer()

```
void Webserv::_initServer (
            vec< config_t > & confs)  [private]
```

Parameters

confs	

Here is the call graph for this function:



Here is the caller graph for this function:

6.26.2.5 _initServerCreate()

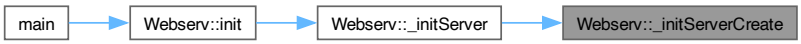
```
void Webserv::_initServerCreate (
            const port_t & port)  [private]
```

Parameters

port	

Here is the call graph for this function:

Here is the caller graph for this function:



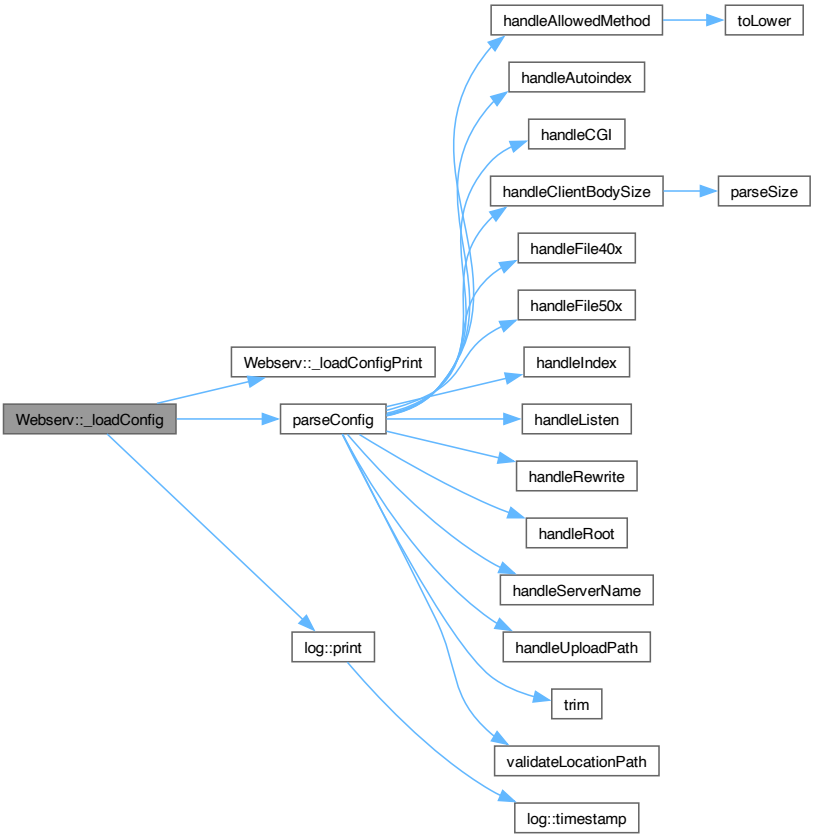
6.26.2.6 _loadConfig()

```
void Webserv::_loadConfig (
            const char * filename,
            vec< config_t > & holder)  [private]
```

Parameters

filename	
holder	

Here is the call graph for this function:



Here is the caller graph for this function:

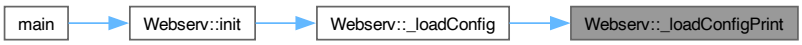
6.26.2.7 _loadConfigPrint()

```
void Webserv::_loadConfigPrint (
            const vec< config_t > & holder) const  [private]
```

Parameters

holder	

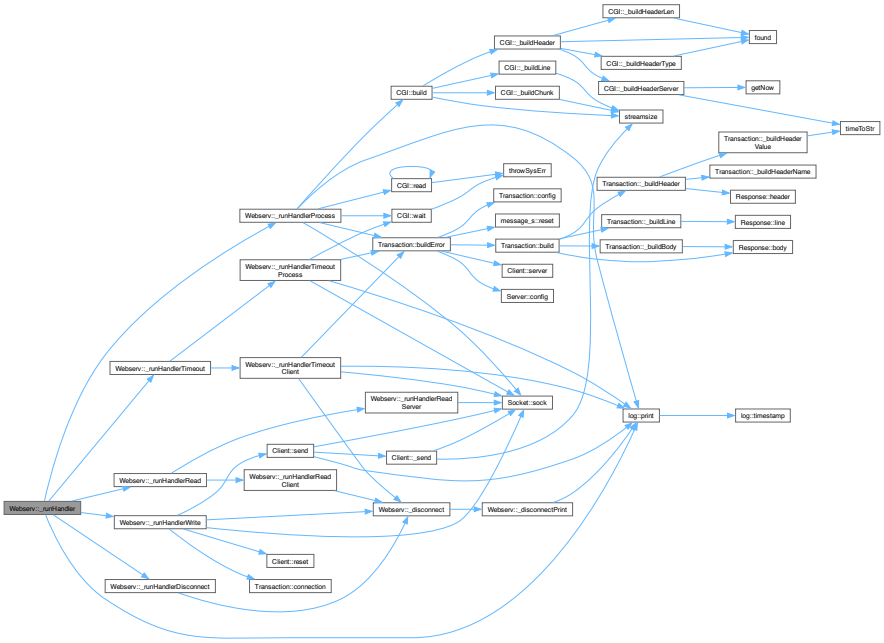
Here is the caller graph for this function:



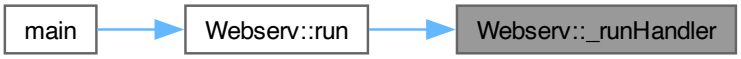
6.26.2.8 _runHandler()

```
void Webserv::_runHandler ()  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.26.2.9 _runHandlerDisconnect()

```
bool Webserv::_runHandlerDisconnect (
            const event_t & ev)  [private]
```

Parameters

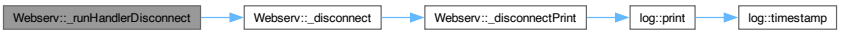
ev	

Returns

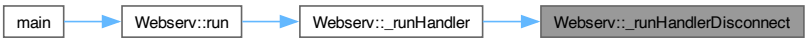
true

false

Here is the call graph for this function:



Here is the caller graph for this function:



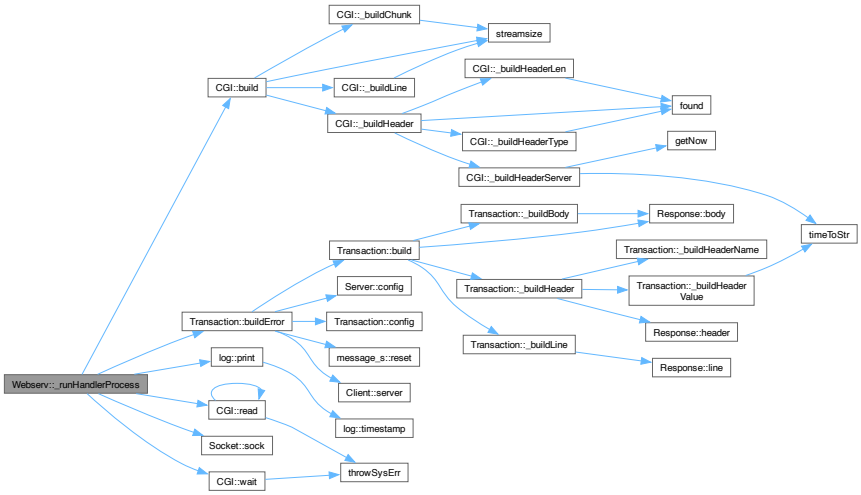
6.26.2.10 _runHandlerProcess()

```
void Webserv::_runHandlerProcess (
            const event_t & ev)  [private]
```

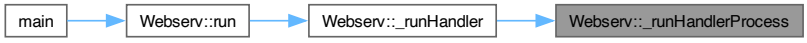
Parameters

ev	

Here is the call graph for this function:



Here is the caller graph for this function:



6.26.2.11 _runHandlerRead()

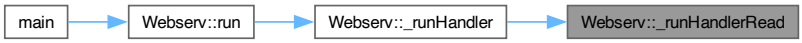
```
void Webserv::_runHandlerRead (
            const event_t & ev)  [private]
```

Parameters

ev	

Here is the call graph for this function:

Here is the caller graph for this function:



6.26.2.12 _runHandlerReadClient()

```
void Webserv::_runHandlerReadClient (
            const event_t & ev)  [private]
```

Parameters

ev	

Here is the call graph for this function:

Here is the caller graph for this function:

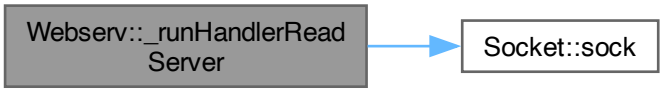
6.26.2.13 _runHandlerReadServer()

```
void Webserv::_runHandlerReadServer (
            const uintptr_t & ident)  [private]
```

Parameters

ident	

Here is the call graph for this function:



Here is the caller graph for this function:

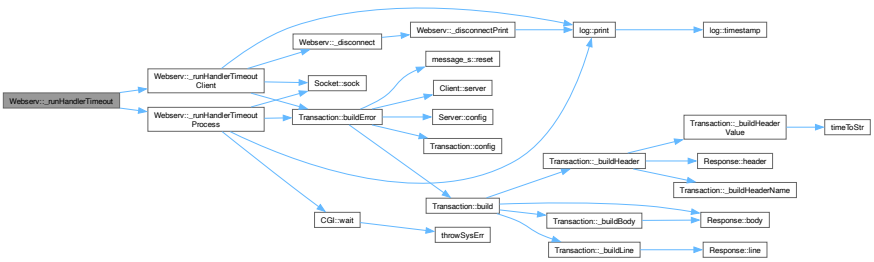
6.26.2.14 _runHandlerTimeout()

```
void Webserv::_runHandlerTimeout (
            const event_t & ev)  [private]
```

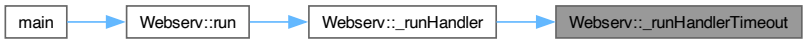
Parameters

ev	

Here is the call graph for this function:



Here is the caller graph for this function:



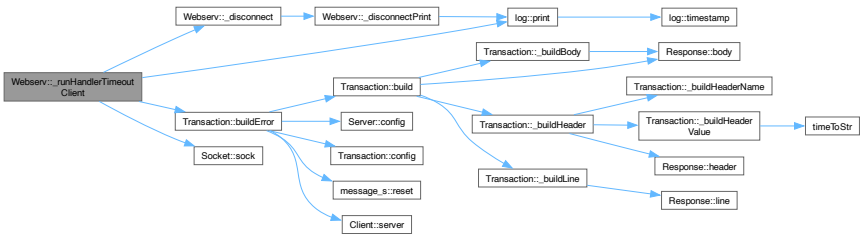
6.26.2.15 _runHandlerTimeoutClient()

```
void Webserv::_runHandlerTimeoutClient (
            const event_t & ev,
            Client & cl)  [private]
```

Parameters

ev	
cl	

Here is the call graph for this function:



Here is the caller graph for this function:

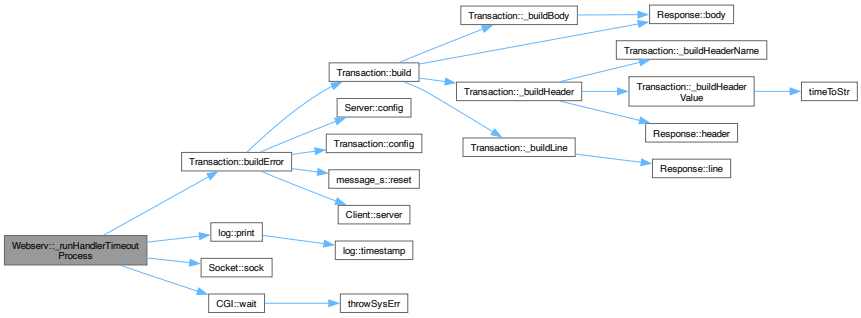
6.26.2.16 _runHandlerTimeoutProcess()

```
void Webserv::_runHandlerTimeoutProcess (
            const event_t & ev)  [private]
```

Parameters

ev	

Here is the call graph for this function:



Here is the caller graph for this function:

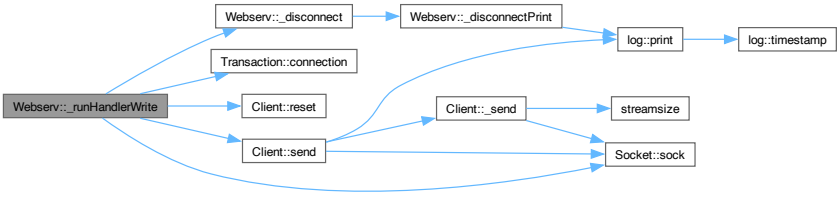
6.26.2.17 _runHandlerWrite()

```
void Webserv::_runHandlerWrite (
            const event_t & ev)  [private]
```

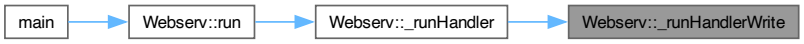
Parameters

ev	

Here is the call graph for this function:



Here is the caller graph for this function:



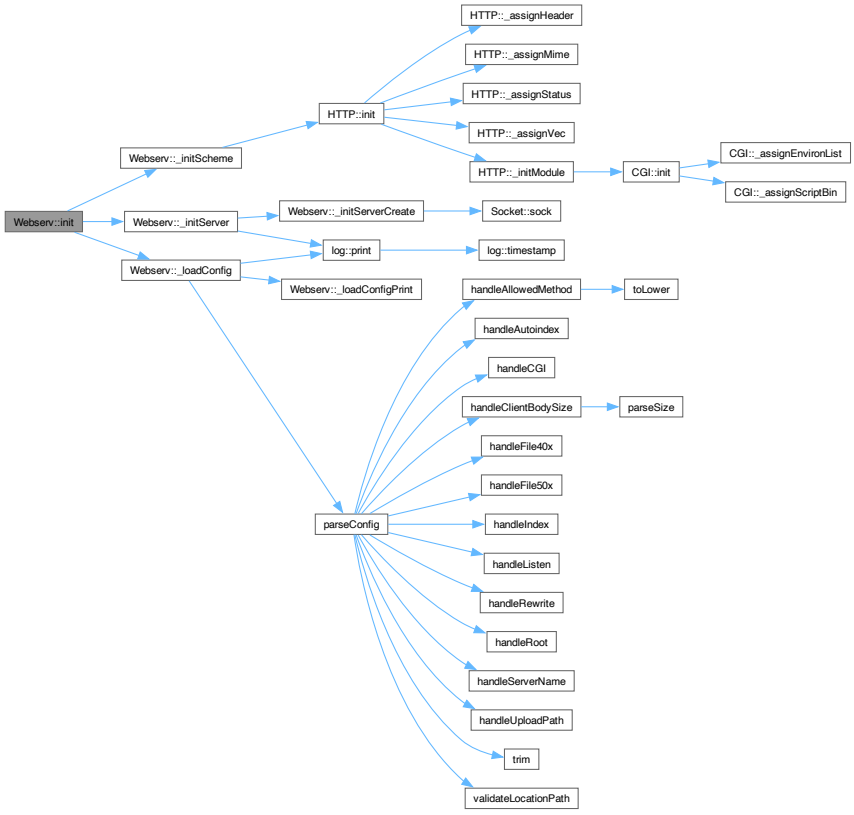
6.26.2.18 init()

```
void Webserv::init (
            const char * filename)
```

Parameters

filename	

Here is the call graph for this function:

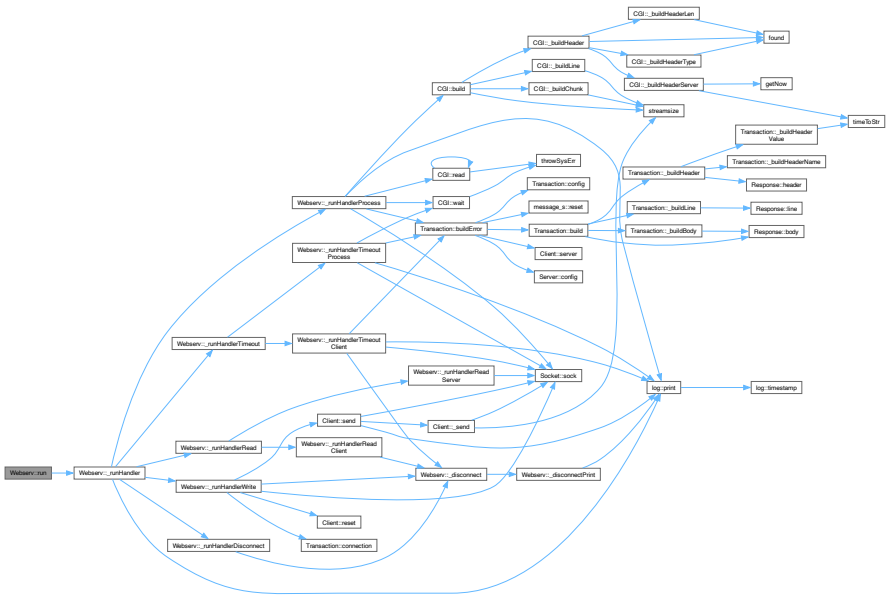


Here is the caller graph for this function:

6.26.2.19 run()

```
void Webserv::run ()
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.26.3 Member Data Documentation

6.26.3.1 _evnt

```
Kqueue& Webserv::_evnt  [private]
```

6.26.3.2 _list

```
struct Webserv::list_s Webserv::_list  [private]
```

6.26.3.3 _map

```
struct Webserv::map_s Webserv::_map  [private]
```

6.26.3.4 state

```
int Webserv::state
```

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

- src/core/Webserv.hpp
- src/core/Webserv.cpp

Chapter 7

File Documentation

7.1 html/donghong/bin/timeout.py File Reference

Namespaces

- namespace timeout

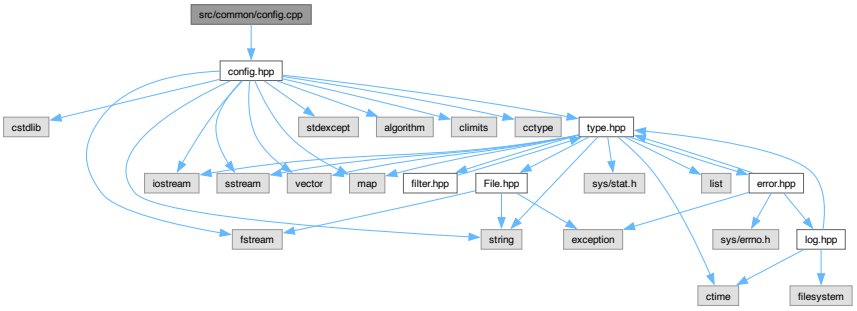
Functions

- timeout.main ()

7.2 src/common/config.cpp File Reference

```
#include "config.hpp"
```

Include dependency graph for config.cpp:



Functions

- std::string toLower (const std::string &str)
- unsigned int parseSize (const std::string &sizeStr)
- void handleListen (std::istringstream &iss, config_t &currentConfig)
- void handleServerName (std::istringstream &iss, config_t &currentConfig)
- void handleClientBodySize (std::istringstream &iss, config_t &currentConfig)
- void handleFile40x (std::istringstream &iss, config_t &currentConfig)
- void handleFile50x (std::istringstream &iss, config_t &currentConfig)
- void handleRoot (std::istringstream &iss, location_t &currentLocation)
- void handleAllowedMethod (std::istringstream &iss, location_t &currentLocation)
- void handleAutoindex (std::istringstream &iss, location_t &currentLocation)
- void handleIndex (std::istringstream &iss, location_t &currentLocation)
- void handleRewrite (std::istringstream &iss, location_t &currentLocation)
- void handleCGI (std::istringstream &iss, location_t &currentLocation)
- void handleUploadPath (std::istringstream &iss, location_t &currentLocation)
- bool validateLocationPath (const std::string &path)
- std::string trim (const std::string &str)
- void parseConfig (std::vector< config_t > &serv, const std::string &filename)

7.2.1 Function Documentation

7.2.1.1 handleAllowedMethod()

```
void handleAllowedMethod (
            std::istringstream & iss,
            location_t & currentLocation)
```

Here is the call graph for this function:

Here is the caller graph for this function:

7.2.1.2 handleAutoindex()

```
void handleAutoindex (
            std::istringstream & iss,
            location_t & currentLocation)
```

Here is the caller graph for this function:

7.2.1.3 handleCGI()

```
void handleCGI (
            std::istringstream & iss,
            location_t & currentLocation)
```

Here is the caller graph for this function:

7.2.1.4 handleClientBodySize()

```
void handleClientBodySize (
            std::istringstream & iss,
            config_t & currentConfig)
```

Here is the call graph for this function:

Here is the caller graph for this function:

7.2.1.5 handleFile40x()

```
void handleFile40x (
            std::istringstream & iss,
            config_t & currentConfig)
```

Here is the caller graph for this function:

7.2.1.6 handleFile50x()

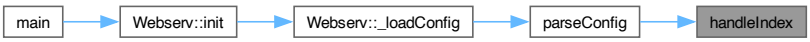
```
void handleFile50x (
            std::istringstream & iss,
            config_t & currentConfig)
```

Here is the caller graph for this function:

7.2.1.7 handleIndex()

```
void handleIndex (
            std::istringstream & iss,
            location_t & currentLocation)
```

Here is the caller graph for this function:



7.2.1.8 handleListen()

```
void handleListen (
            std::istringstream & iss,
            config_t & currentConfig)
```

Here is the caller graph for this function:

7.2.1.9 handleRewrite()

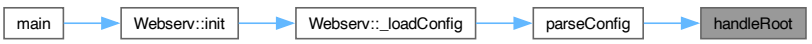
```
void handleRewrite (
            std::istringstream & iss,
            location_t & currentLocation)
```

Here is the caller graph for this function:

7.2.1.10 handleRoot()

```
void handleRoot (
            std::istringstream & iss,
            location_t & currentLocation)
```

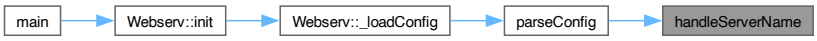
Here is the caller graph for this function:



7.2.1.11 handleServerName()

```
void handleServerName (
            std::istringstream & iss,
            config_t & currentConfig)
```

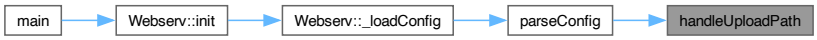
Here is the caller graph for this function:



7.2.1.12 handleUploadPath()

```
void handleUploadPath (
            std::istringstream & iss,
            location_t & currentLocation)
```

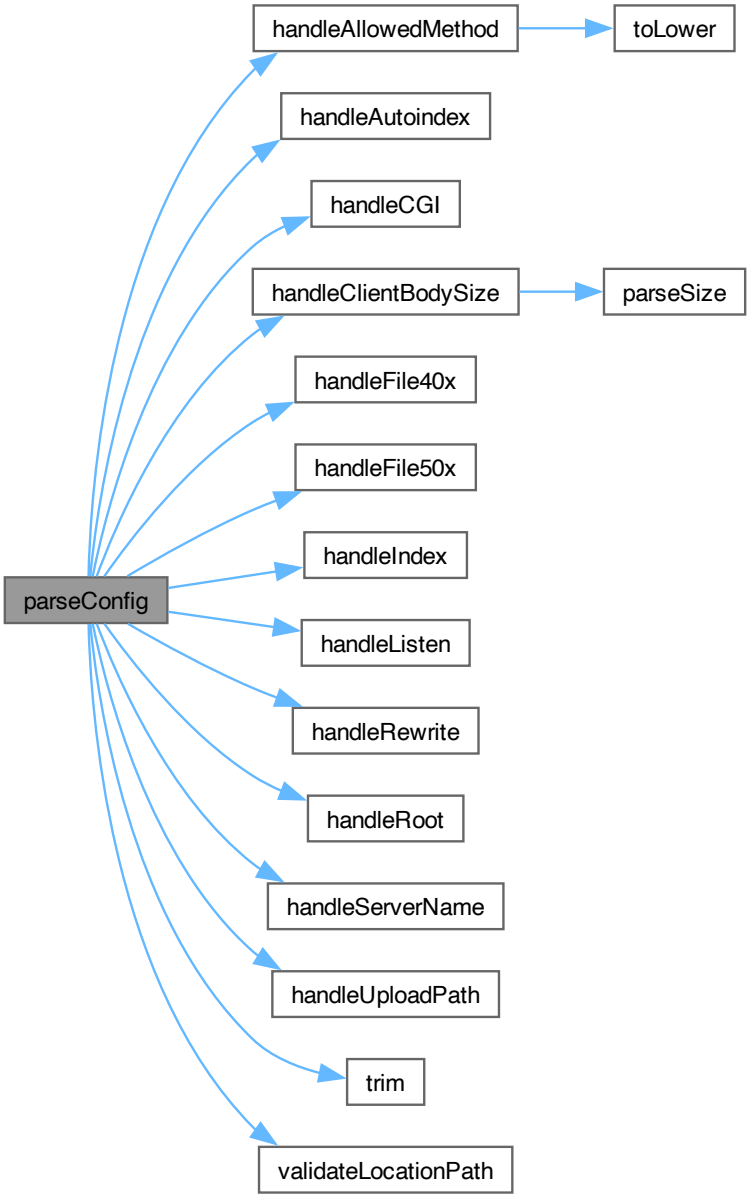
Here is the caller graph for this function:



7.2.1.13 parseConfig()

```
void parseConfig (
            std::vector< config_t > & serv,
            const std::string & filename)
```

Here is the call graph for this function:



Here is the caller graph for this function:

7.2.1.14 parseSize()

```
unsigned int parseSize (
            const std::string & sizeStr)
```

Here is the caller graph for this function:

7.2.1.15 toLower()

```
std::string toLower (
            const std::string & str)
```

Here is the caller graph for this function:

7.2.1.16 trim()

```
std::string trim (
            const std::string & str)
```

Here is the caller graph for this function:

7.2.1.17 validateLocationPath()

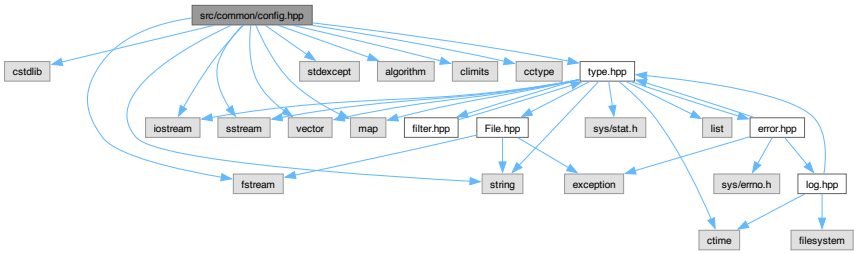
```
bool validateLocationPath (
            const std::string & path)
```

Here is the caller graph for this function:

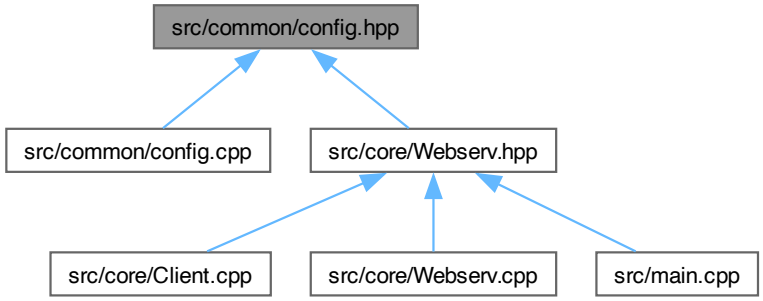
7.3 src/common/config.hpp File Reference

```
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <map>
#include <stdexcept>
#include <algorithm>
#include <climits>
#include <cctype>
#include "type.hpp"
```

Include dependency graph for config.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define on true
- #define off false

Functions

- void parseConfig (std::vector< config_t > &serv, const std::string &filename)

7.3.1 Macro Definition Documentation

7.3.1.1 off

```
#define off false
```

7.3.1.2 on

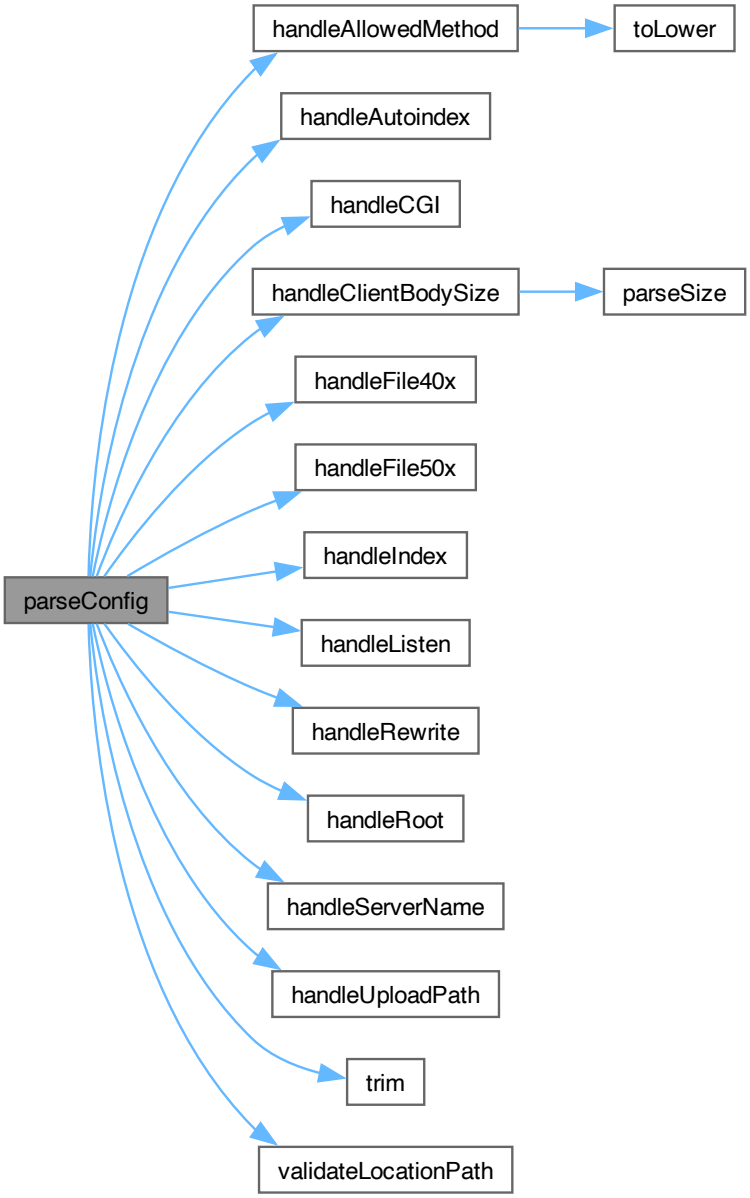
```
#define on true
```

7.3.2 Function Documentation

7.3.2.1 parseConfig()

```
void parseConfig (
            std::vector< config_t > & serv,
            const std::string & filename)
```

Here is the call graph for this function:



Here is the caller graph for this function:

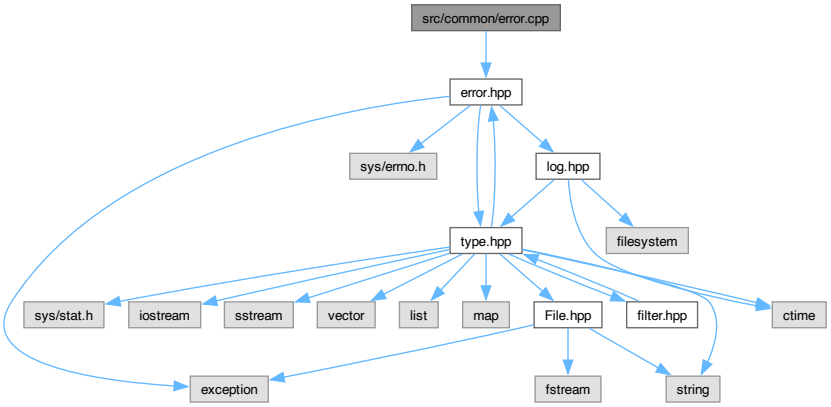
7.4 config.hpp

Go to the documentation of this file.

```
00001 #ifndef CONFIG_HPP
00002 # define CONFIG_HPP
00003
00004 #include <cstdlib>
00005 #include <fstream>
00006 #include <iostream>
00007 #include <sstream>
00008 #include <string>
00009 #include <vector>
00010 #include <map>
00011 #include <stdexcept>
00012 #include <algorithm>
00013 #include <climits>
00014 #include <cctype>
00015
00016 #include "type.hpp"
00017
00018 #define on true
00019 #define off false
00020
00021 void parseConfig(std::vector<config_t>& serv, const std::string& filename);
00022
00023 #endif
```

7.5 src/common/error.cpp File Reference

`#include "error.hpp"`
Include dependency graph for error.cpp:



Functions

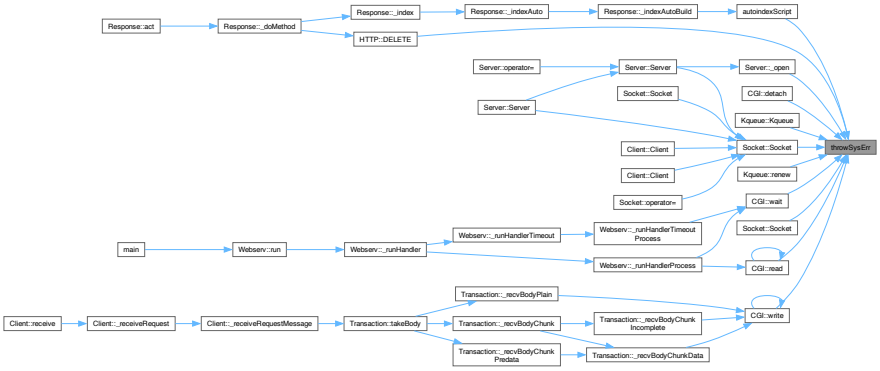
- void throwSysErr (const str_t &fname)
- void throwSysErr (const str_t &fname, const uint_t &code)
- void throwSysErr (const str_t &fname, const uint_t &code, const size_t &confidx)

7.5.1 Function Documentation

7.5.1.1 throwSysErr() [1/3]

```
void throwSysErr (
            const str_t & fname)
```

Here is the caller graph for this function:



7.5.1.2 throwSysErr() [2/3]

```
void throwSysErr (
            const str_t & fname,
            const uint_t & code)
```

7.5.1.3 throwSysErr() [3/3]

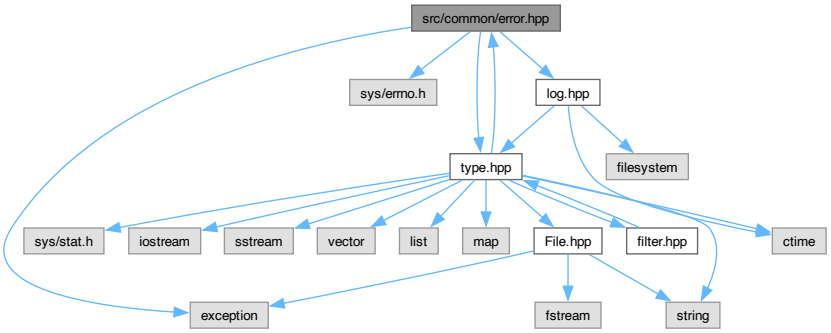
```
void throwSysErr (
            const str_t & fname,
            const uint_t & code,
            const size_t & confidx)
```

7.6 src/common/error.hpp File Reference

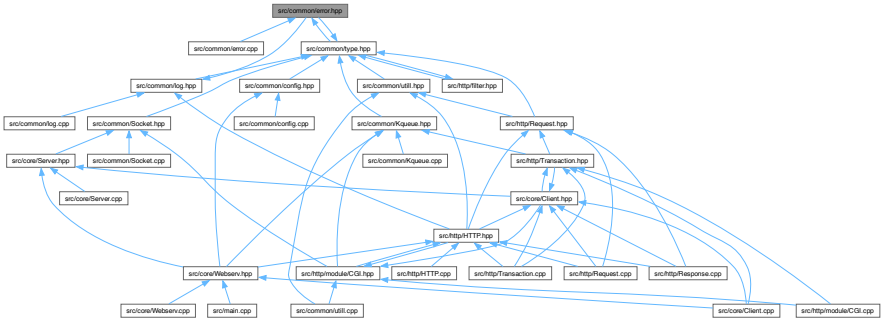
```
#include <exception>
#include <sys/errno.h>
#include "type.hpp"
```

```
#include "log.hpp"
```

Include dependency graph for error.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct errstat_s

Macros

- #define ERROR -1
- #define SUCCESS 0

Typedefs

- typedef int errno_t
- typedef std::runtime_error err_t
- typedef std::exception exception_t
- typedef errstat_s errstat_t

Enumerations

- enum err_msg_e {
  TOKEN_FAIL_GETLINE , INVALID_REQUEST_LINE , INVALID_REQUEST_FIELD , VERSION_NOT_SUPPORTED ,
  CHUNK_EXCEED_HEX , TE_WITH_CONTENT_LEN , TE_NOT_IMPLEMENTED , SOURCE_NOT_FOUND ,
  SOURCE_NOT_DIR , CGI_WITH_NOT_ALLOWED , GET_WITH_BODY , POST_EMPTY_CONTENT_LEN ,
  POST_OVER_CONTENT_LEN , CGI_EXIT_FAILURE , CGI_EXCEED_TIME , FAIL_SEND }

Functions

- void throwSysErr (const str_t &)
- void throwSysErr (const str_t &, const uint_t &)
- void throwSysErr (const str_t &, const uint_t &, const size_t &)

Variables

- int errno
- const str_t err_msg [ ]

7.6.1 Macro Definition Documentation

7.6.1.1 ERROR

```
#define ERROR -1
```

7.6.1.2 SUCCESS

```
#define SUCCESS 0
```

7.6.2 Typedef Documentation

7.6.2.1 err_t

```
typedef std::runtime_error err_t
```

7.6.2.2 errno_t

```
typedef int errno_t
```

7.6.2.3 errstat_t

```
typedef errstat_s errstat_t
```

7.6.2.4 exception_t

```
typedef std::exception exception_t
```

7.6.3 Enumeration Type Documentation

7.6.3.1 err_msg_e

```
enum err_msg_e
```

Enumerator

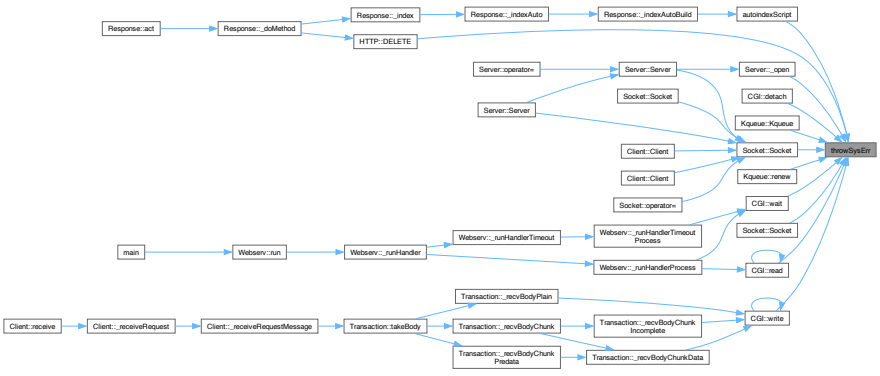
TOKEN_FAIL_GETLINE	
INVALID_REQUEST_LINE	
INVALID_REQUEST_FIELD	
VERSION_NOT_SUPPORTED	
CHUNK_EXCEED_HEX	
TE_WITH_CONTENT_LEN	
TE_NOT_IMPLEMENTED	
SOURCE_NOT_FOUND	
SOURCE_NOT_DIR	
CGI_WITH_NOT_ALLOWED	
GET_WITH_BODY	
POST_EMPTY_CONTENT_LEN	
POST_OVER_CONTENT_LEN	
CGI_EXIT_FAILURE	
CGI_EXCEED_TIME	
FAIL_SEND	

7.6.4 Function Documentation

7.6.4.1 throwSysErr() [1/3]

```
void throwSysErr (
            const str_t & fname)
```

Here is the caller graph for this function:



7.6.4.2 throwSysErr() [2/3]

```
void throwSysErr (
            const str_t & fname,
            const uint_t & code)
```

7.6.4.3 throwSysErr() [3/3]

```
void throwSysErr (
              const str_t & fname,
              const uint_t & code,
              const size_t & confidx)
```

7.6.5 Variable Documentation

7.6.5.1 err_msg

```
const str_t err_msg[]
```

Initial value:

```
= {
    "token fail to getline",

    "invalid request line",
    "invalid request field",
    "version not supported",
    "a cnunked data exceeding hexsize",

    "the transfer_encoding and content_length can not be taken at the same time",
    "requested Transfer-Encoding way is not implemented",
    "target source is not exist",
    "requested with slash end but target source is not dir",
    "cgi may be with GET and POST method only",

    "the GET request may not be with body",
    "the requested body size is unknown from client request",
    "the requested body size exceeds configured size of limitation",

    "the CGI failed to exit as SUCCESS",
    "the CGI exceed time to limit for proceeding",

    "fail to send"
}
```

7.6.5.2 errno

```
int errno  [extern]
```

7.7 error.hpp

Go to the documentation of this file.

```
00001 #ifndef ERROR_HPP
00002 # define ERROR_HPP
00003
00004 # include <exception>
00005 # include <sys/errno.h>
00006
00007 # include "type.hpp"
00008
00009 # define ERROR -1
00010 # define SUCCESS 0
00011
00012 extern int errno;
00013
00014 typedef int                 errno_t;
00015 typedef std::runtime_error  err_t;
00016 typedef std::exception      exception_t;
00017
00018 enum err_msg_e {
00019     TOKEN_FAIL_GETLINE,
00020
```

```
00021     INVALID_REQUEST_LINE,
00022     INVALID_REQUEST_FIELD,
00023     VERSION_NOT_SUPPORTED,
00024     CHUNK_EXCEED_HEX,
00025
00026     TE_WITH_CONTENT_LEN,
00027     TE_NOT_IMPLEMENTED,
00028     SOURCE_NOT_FOUND,
00029     SOURCE_NOT_DIR,
00030     CGI_WITH_NOT_ALLOWED,
00031
00032     GET_WITH_BODY,
00033     POST_EMPTY_CONTENT_LEN,
00034     POST_OVER_CONTENT_LEN,
00035
00036     CGI_EXIT_FAILURE,
00037     CGI_EXCEED_TIME,
00038
00039     FAIL_SEND
00040 };
00041
00042 const str_t    err_msg[] = {
00043     "token fail to getline",
00044
00045     "invalid request line",
00046     "invalid request field",
00047     "version not supported",
00048     "a cnunked data exceeding hexsize",
00049
00050     "the transfer_encoding and content_length can not be taken at the same time",
00051     "requested Transfer-Encoding way is not implemented",
00052     "target source is not exist",
00053     "requested with slash end but target source is not dir",
00054     "cgi may be with GET and POST method only",
00055
00056     "the GET request may not be with body",
00057     "the requested body size is unknown from client request",
00058     "the requested body size exceeds configured size of limitation",
00059
00060     "the CGI failed to exit as SUCCESS",
00061     "the CGI exceed time to limit for proceeding",
00062
00063     "fail to send"
00064 };
00065
00066 typedef struct errstat_s: err_t {
00067     uint_t        code;
00068     size_t      confidx;
00069
00070     errstat_s( const uint_t& );
00071     errstat_s( const uint_t&, const size_t& );
00072
00073     errstat_s( const uint_t&, const str_t& );
00074     errstat_s( const uint_t&, const str_t&, const size_t& );
00075
00076 }   errstat_t;
00077
00078 void    throwSysErr( const str_t&);
00079 void    throwSysErr( const str_t&, const uint_t& );
00080 void    throwSysErr( const str_t&, const uint_t&, const size_t& );
00081
00082 # include "log.hpp"
00083
00084 #endif
00085
00086 /*
00087 15.5.1. 400 Bad Request
00088 The 400 (Bad Request) status code indicates that the server cannot or will not
00089 process the request due to something that is perceived to be a client error
00090 (e.g., malformed request syntax, invalid request message framing, or deceptive
00091 request routing).
00092
00093 15.5.4. 403 Forbidden
00094 The 403 (Forbidden) status code indicates that the server understood the request
00095 but refuses to fulfill it. A server that wishes to make public why the request
00096 has been forbidden can describe that reason in the response content (if any).
00097
00098 If authentication credentials were provided in the request, the server considers
00099 them insufficient to grant access. The client SHOULD NOT automatically repeat
00100 the request with the same credentials. The client MAY repeat the request with
00101 new or different credentials. However, a request might be forbidden for reasons
00102 unrelated to the credentials.
00103
00104 An origin server that wishes to "hide" the current existence of a forbidden
00105 target resource MAY instead respond with a status code of 404 (Not Found).
00106
00107 15.5.5. 404 Not Found
```

```
00108 The 404 (Not Found) status code indicates that the origin server did not find a
00109 current representation for the target resource or is not willing to disclose
00110 that one exists. A 404 status code does not indicate whether this lack of
00111 representation is temporary or permanent; the 410 (Gone) status code is
00112 preferred over 404 if the origin server knows, presumably through some
00113 configurable means, that the condition is likely to be permanent.
00114
00115 A 404 response is heuristically cacheable; i.e., unless otherwise indicated by
00116 the method definition or explicit cache controls (see Section 4.2.2 of
00117 [CACHING]).
00118
00119 15.5.6. 405 Method Not Allowed
00120 The 405 (Method Not Allowed) status code indicates that the method received in
00121 the request-line is known by the origin server but not supported by the target
00122 resource. The origin server MUST generate an Allow header field in a 405
00123 response containing a list of the target resource's currently supported methods.
00124
00125 A 405 response is heuristically cacheable; i.e., unless otherwise indicated by
00126 the method definition or explicit cache controls (see Section 4.2.2 of
00127 [CACHING]).
00128
00129 15.5.7. 406 Not Acceptable
00130 The 406 (Not Acceptable) status code indicates that the target resource does not
00131 have a current representation that would be acceptable to the user agent,
00132 according to the proactive negotiation header fields received in the request
00133 (Section 12.1), and the server is unwilling to supply a default representation.
00134
00135 The server SHOULD generate content containing a list of available representation
00136 characteristics and corresponding resource identifiers from which the user or
00137 user agent can choose the one most appropriate. A user agent MAY automatically
00138 select the most appropriate choice from that list. However, this specification
00139 does not define any standard for such automatic selection, as described in
00140 Section 15.4.1.
00141
00142 15.5.9. 408 Request Timeout
00143 The 408 (Request Timeout) status code indicates that the server did not receive
00144 a complete request message within the time that it was prepared to wait.
00145
00146 If the client has an outstanding request in transit, it MAY repeat that request.
00147 If the current connection is not usable (e.g., as it would be in HTTP/1.1
00148 because request delimitation is lost), a new connection will be used.
00149
00150 15.5.10. 409 Conflict
00151 The 409 (Conflict) status code indicates that the request could not be completed
00152 due to a conflict with the current state of the target resource. This code is
00153 used in situations where the user might be able to resolve the conflict and
00154 resubmit the request. The server SHOULD generate content that includes enough
00155 information for a user to recognize the source of the conflict.
00156
00157 Conflicts are most likely to occur in response to a PUT request. For example, if
00158 versioning were being used and the representation being PUT included changes to
00159 a resource that conflict with those made by an earlier (third-party) request,
00160 the origin server might use a 409 response to indicate that it can't complete
00161 the request. In this case, the response representation would likely contain
00162 information useful for merging the differences based on the revision log::history.
00163
00164 15.5.11. 410 Gone
00165 The 410 (Gone) status code indicates that access to the target resource is no
00166 longer available at the origin server and that this condition is likely to be
00167 permanent. If the origin server does not know, or has no facility to determine,
00168 whether or not the condition is permanent, the status code 404 (Not Found) ought
00169 to be used instead.
00170
00171 The 410 response is primarily intended to assist the task of web maintenance by
00172 notifying the recipient that the resource is intentionally unavailable and that
00173 the server owners desire that remote links to that resource be removed. Such an
00174 event is common for limited-time, promotional services and for resources
00175 belonging to individuals no longer associated with the origin server's site. It
00176 is not necessary to mark all permanently unavailable resources as "gone" or to
00177 keep the mark for any length of time -- that is left to the discretion of the
00178 server owner.
00179
00180 A 410 response is heuristically cacheable; i.e., unless otherwise indicated by
00181 the method definition or explicit cache controls (see Section 4.2.2 of
00182 [CACHING]).
00183
00184 15.5.12. 411 Length Required
00185 The 411 (Length Required) status code indicates that the server refuses to
00186 accept the request without a defined Content-Length (Section 8.6). The client
00187 MAY repeat the request if it adds a valid Content-Length header field containing
00188 the length of the request content.
00189
00190 15.5.13. 412 Precondition Failed
00191 The 412 (Precondition Failed) status code indicates that one or more conditions
00192 given in the request header fields evaluated to false when tested on the server
00193 (Section 13). This response status code allows the client to place preconditions
00194 on the current resource state (its current representations and metadata) and,
```

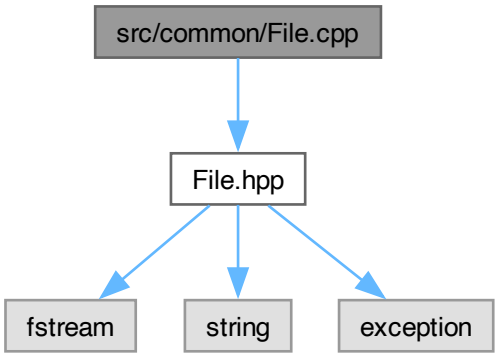
```
00195 thus, prevent the request method from being applied if the target resource is in
00196 an unexpected state.
00197
00198 15.5.14. 413 Content Too Large
00199 The 413 (Content Too Large) status code indicates that the server is refusing to
00200 process a request because the request content is larger than the server is
00201 willing or able to process. The server MAY terminate the request, if the
00202 protocol version in use allows it; otherwise, the server MAY close the
00203 connection.
00204
00205 If the condition is temporary, the server SHOULD generate a Retry-After header
00206 field to indicate that it is temporary and after what time the client MAY try
00207 again.
00208
00209 15.5.15. 414 URI Too Long
00210 The 414 (URI Too Long) status code indicates that the server is refusing to
00211 service the request because the target URI is longer than the server is willing
00212 to interpret. This rare condition is only likely to occur when a client has
00213 improperly converted a POST request to a GET request with long query
00214 information, when the client has descended into an infinite loop of redirection
00215 (e.g., a redirected URI prefix that points to a suffix of itself) or when the
00216 server is under attack by a client attempting to exploit potential security
00217 holes.
00218
00219 A 414 response is heuristically cacheable; i.e., unless otherwise indicated by
00220 the method definition or explicit cache controls (see Section 4.2.2 of
00221 [CACHING]).
00222
00223 15.5.16. 415 Unsupported Media Type
00224 The 415 (Unsupported Media Type) status code indicates that the origin server is
00225 refusing to service the request because the content is in a format not supported
00226 by this method on the target resource.
00227
00228 The format problem might be due to the request's indicated Content-Type or
00229 Content-Encoding, or as a result of inspecting the data directly.
00230
00231 If the problem was caused by an unsupported content coding, the Accept-Encoding
00232 response header field (Section 12.5.3) ought to be used to indicate which (if
00233 any) content codings would have been accepted in the request.
00234
00235 On the other hand, if the cause was an unsupported media type, the Accept
00236 response header field (Section 12.5.1) can be used to indicate which media types
00237 would have been accepted in the request.
00238
00239 15.5.17. 416 Range Not Satisfiable
00240 The 416 (Range Not Satisfiable) status code indicates that the set of ranges in
00241 the request's Range header field (Section 14.2) has been rejected either because
00242 none of the requested ranges are satisfiable or because the client has requested
00243 an excessive number of small or overlapping ranges (a potential denial of
00244 service attack).
00245
00246 Each range unit defines what is required for its own range sets to be
00247 satisfiable. For example, Section 14.1.2 defines what makes a bytes range set
00248 satisfiable.
00249
00250 A server that generates a 416 response to a byte-range request SHOULD generate a
00251 Content-Range header field specifying the current length of the selected
00252 representation (Section 14.4).
00253
00254 For example:
00255
00256 HTTP/1.1 416 Range Not Satisfiable
00257 Date: Fri, 20 Jan 2012 15:41:54 GMT
00258 Content-Range: bytes /47022
00259 Note: Because servers are free to ignore Range, many implementations will
00260 respond with the entire selected representation in a 200 (OK) response. That is
00261 partly because most clients are prepared to receive a 200 (OK) to complete the
00262 task (albeit less efficiently) and partly because clients might not stop making
00263 an invalid range request until they have received a complete representation.
00264 Thus, clients cannot depend on receiving a 416 (Range Not Satisfiable) response
00265 even when it is most appropriate.
00266
00267 15.5.18. 417 Expectation Failed
00268 The 417 (Expectation Failed) status code indicates that the expectation given in
00269 the request's Expect header field (Section 10.1.1) could not be met by at least
00270 one of the inbound servers.
00271
00272 15.5.19. 418 (Unused)
00273 [RFC2324] was an April 1 RFC that lampooned the various ways HTTP was abused;
00274 one such abuse was the definition of an application-specific 418 status code,
00275 which has been deployed as a joke often enough for the code to be unusable for
00276 any future use.
00277
00278 Therefore, the 418 status code is reserved in the IANA HTTP Status Code
00279 Registry. This indicates that the status code cannot be assigned to other
00280 applications currently. If future circumstances require its use (e.g.,
00281 exhaustion of 4NN status codes), it can be re-assigned to another use.
```

```
00282
00283 15.5.20. 421 Misdirected Request
00284 The 421 (Misdirected Request) status code indicates that the request was
00285 directed at a server that is unable or unwilling to produce an authoritative
00286 response for the target URI. An origin server (or gateway acting on behalf of
00287 the origin server) sends 421 to reject a target URI that does not match an
00288 origin for which the server has been configured (Section 4.3.1) or does not
00289 match the connection context over which the request was received (Section 7.4).
00290
00291 A client that receives a 421 (Misdirected Request) response MAY retry the
00292 request, whether or not the request method is idempotent, over a different
00293 connection, such as a fresh connection specific to the target resource's origin,
00294 or via an alternative service [ALTSVC].
00295
00296 A proxy MUST NOT generate a 421 response.
00297
00298 15.5.21. 422 Unprocessable Content
00299 The 422 (Unprocessable Content) status code indicates that the server
00300 understands the content type of the request content (hence a 415 (Unsupported
00301 Media Type) status code is inappropriate), and the syntax of the request content
00302 is correct, but it was unable to process the contained instructions. For
00303 example, this status code can be sent if an XML request content contains
00304 well-formed (i.e., syntactically correct), but semantically erroneous XML
00305 instructions.
00306
00307 15.5.22. 426 Upgrade Required
00308 The 426 (Upgrade Required) status code indicates that the server refuses to
00309 perform the request using the current protocol but might be willing to do so
00310 after the client upgrades to a different protocol. The server MUST send an
00311 Upgrade header field in a 426 response to indicate the required protocol(s)
00312 (Section 7.8).
00313
00314 Example:
00315
00316 HTTP/1.1 426 Upgrade Required
00317 Upgrade: HTTP/3.0
00318 Connection: Upgrade
00319 Content-Length: 53
00320 Content-Type: text/plain
00321
00322 This service requires use of the HTTP/3.0 protocol.
00323 */
```

7.8 src/common/File.cpp File Reference

`#include "File.hpp"`

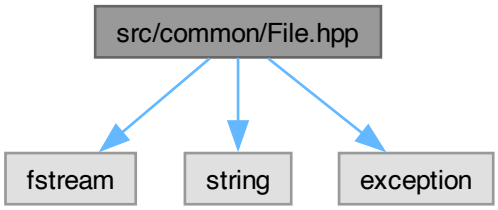
Include dependency graph for File.cpp:



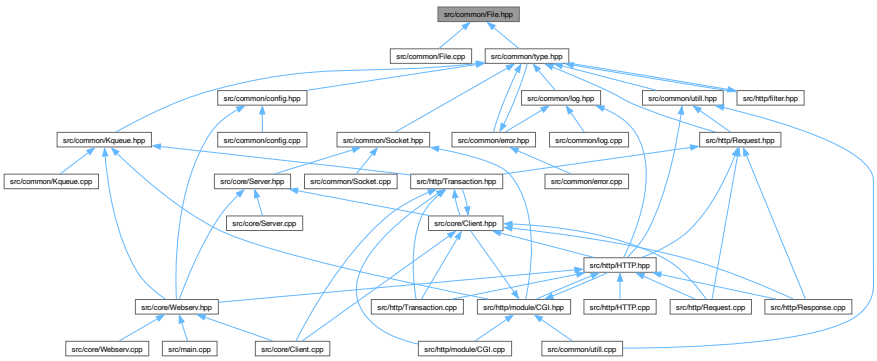
7.9 src/common/File.hpp File Reference

```
#include <fstream>
#include <string>
#include <exception>
```

Include dependency graph for File.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class File

Enumerations

- enum Mode { READ , READ_BINARY , WRITE , WRITE_APP }

7.9.1 Enumeration Type Documentation

7.9.1.1 Mode

```
enum Mode
```

Enumerator

READ	
READ_BINARY	
WRITE	
WRITE_APP	

7.10 File.hpp

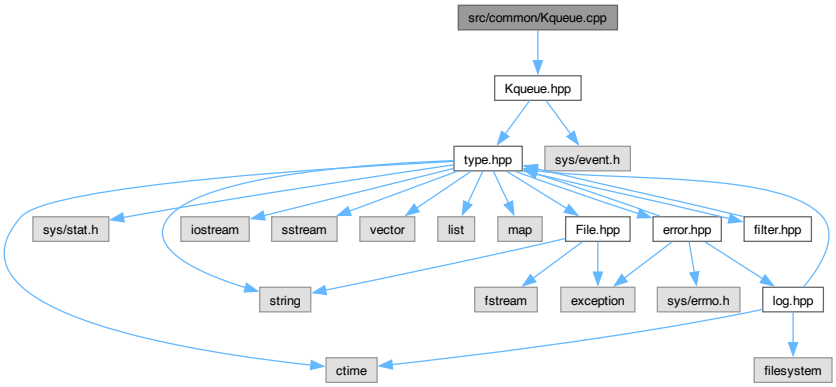
Go to the documentation of this file.

```
00001 #ifndef FILE_HPP
00002 # define FILE_HPP
00003
00004 # include <fstream>
00005
00006 # include <string>
00007 # include <exception>
00008
00009 /*
00010     app     (append) Set the stream's position indicator to the end of the stream before each output
      operation.
00011     ate     (at end) Set the stream's position indicator to the end of the stream on opening.
00012     binary  (binary) Consider stream as binary rather than text.
00013     in      (input) Allow input operations on the stream.
00014     out     (output) Allow output operations on the stream.
00015     trunc   (truncate) Any current content is discarded, assuming a length of zero on opening.
00016 */
00017
00018 enum Mode { READ, READ_BINARY, WRITE, WRITE_APP };
00019
00020 class File {
00021     public:
00022         std::fstream    fs;
00023
00024         File( const std::string&, int );
00025         ~File( void );
00026
00027     private:
00028         File( void );
00029         File( const File& );
00030         File&   operator=( const File& );
00031
00032 };
00033
00034 #endif
```

7.11 src/common/Kqueue.cpp File Reference

```
#include "Kqueue.hpp"
```

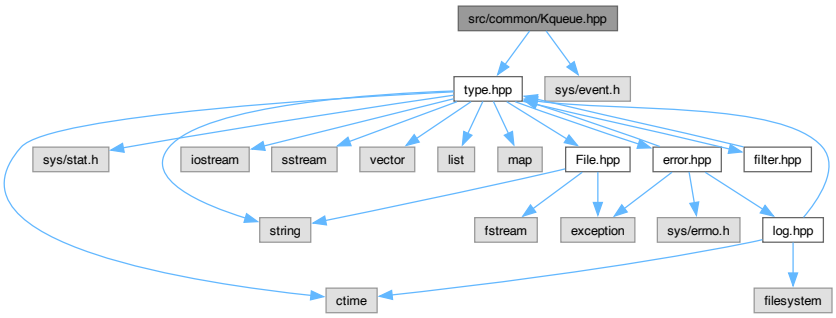
Include dependency graph for Kqueue.cpp:



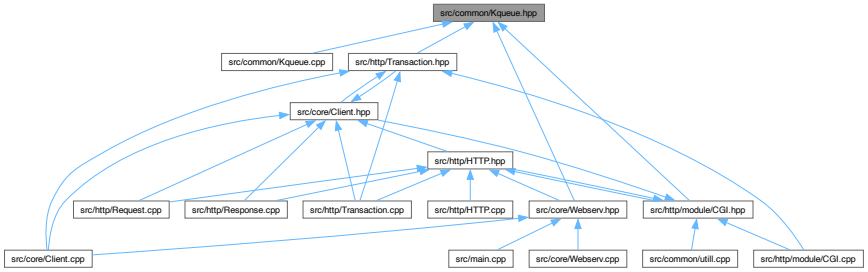
7.12 src/common/Kqueue.hpp File Reference

```
#include "type.hpp"
#include <sys/event.h>
```

Include dependency graph for Kqueue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Kqueue

Macros

- #define TIMEOUT_SEC 120
- #define EVENT_POOL 100000

Typedefs

- typedef struct kevent event_t

7.12.1 Macro Definition Documentation

7.12.1.1 EVENT_POOL

```
#define EVENT_POOL 100000
```

7.12.1.2 TIMEOUT_SEC

```
#define TIMEOUT_SEC 120
```

7.12.2 Typedef Documentation

7.12.2.1 event_t

```
typedef struct kevent event_t
```

7.13 Kqueue.hpp

Go to the documentation of this file.

```
00001 #ifndef KQUEUE_HPP
00002 # define KQUEUE_HPP
00003
00004 # include "type.hpp"
00005
00006 # include <sys/event.h>
00007 typedef struct kevent event_t;
00008
00009 # define TIMEOUT_SEC 120
00010 # define EVENT_POOL 100000
00011
00012 class Kqueue {
00013
00014     public:
00015         Kqueue();
00016         virtual ~Kqueue();
00017
00018         int             fd() const;
00019         const event_t&   que(const size_t&) const;
00020
00021         int             renew();
00022         void            set(uintptr_t, int16_t, uint16_t, uint32_t, intptr_t, void*);
```

```
00023
00024         void*           cast(const int&) const;
00025         int             cast(void*) const;
00026
00027     private:
00028         int             _fd;
00029         vec<event_t>    _que;
00030         struct timespec _timeout;
00031 };
00032
00033 #endif
00034
00035 /*
00036     KQUEUE(2)                     System Calls Manual                    KQUEUE(2)
00037
00038     NAME
00039         kqueue, kevent, kevent64 and kevent_qos - kernel event notification
00040         mechanism
00041
00042     LIBRARY
00043         Standard C Library (libc, -lc)
00044
00045     SYNOPSIS
00046         #include <sys/types.h>
00047         #include <sys/event.h>
00048         #include <sys/time.h>
00049
00050         int
00051         kqueue(void);
00052
00053         int
00054         kevent(int kq, const struct kevent *changelist, int nchanges,
00055             struct kevent *eventlist, int nevents,
00056             const struct timespec *timeout);
00057
00058         int
00059         kevent64(int kq, const struct kevent64_s *changelist, int nchanges,
00060             struct kevent64_s *eventlist, int nevents, unsigned int flags,
00061             const struct timespec *timeout);
00062
00063         int
00064         kevent_qos(int kq, const struct kevent_qos_s *changelist, int nchanges,
00065             struct kevent_qos_s *eventlist, int nevents, void *data_out,
00066             size_t *data_available, unsigned int flags);
00067
00068         EV_SET(&kev, ident, filter, flags, fflags, data, udata);
00069
00070         EV_SET64(&kev, ident, filter, flags, fflags, data, udata, ext[0],
00071             ext[1]);
00072
00073         EV_SET_QOS(&kev, ident, filter, flags, qos, udata, fflags, xflags, data,
00074             ext[0], ext[1], ext[2], ext[3]);
00075
00076     DESCRIPTION
00077         The kqueue() system call allocates a kqueue file descriptor.  This file
00078         descriptor provides a generic method of notifying the user when a kernel
00079         event (kevent) happens or a condition holds, based on the results of
00080         small pieces of kernel code termed filters.
00081
00082         A kevent is identified by an (ident, filter, and optional udata value)
00083         tuple.  It specifies the interesting conditions to be notified about for
00084         that tuple. An (ident, filter, and optional udata value) tuple can only
00085         appear once in a given kqueue.  Subsequent attempts to register the same
00086         tuple for a given kqueue will result in the replacement of the conditions
00087         being watched, not an addition.  Whether the udata value is considered as
00088         part of the tuple is controlled by the EV_UDATA_SPECIFIC flag on the
00089         kevent.
00090
00091         The filter identified in a kevent is executed upon the initial
00092         registration of that event in order to detect whether a preexisting
00093         condition is present, and is also executed whenever an event is passed to
00094         the filter for evaluation.  If the filter determines that the condition
00095         should be reported, then the kevent is placed on the kqueue for the user
00096         to retrieve.
00097
00098         The filter is also run when the user attempts to retrieve the kevent from
00099         the kqueue.  If the filter indicates that the condition that triggered
00100         the event no longer holds, the kevent is removed from the kqueue and is
00101         not returned.
00102
00103         Multiple events which trigger the filter do not result in multiple
00104         kevents being placed on the kqueue; instead, the filter will aggregate
00105         the events into a single struct kevent.  Calling close() on a file
00106         descriptor will remove any kevents that reference the descriptor.
00107
00108         The kqueue() system call creates a new kernel event queue and returns a
00109         descriptor.  The queue is not inherited by a child created with fork(2).
```

```
00110
00111         The kevent,() kevent64() and kevent_qos() system calls are used to
00112         register events with the queue, and return any pending events to the
00113         user.  The changelist argument is a pointer to an array of kevent,
00114         kevent64_s or kevent_qos_s structures, as defined in <sys/event.h>.  All
00115         changes contained in the changelist are applied before any pending events
00116         are read from the queue.  The nchanges argument gives the size of
00117         changelist.
00118
00119         The eventlist argument is a pointer to an array of out kevent, kevent64_s
00120         or kevent_qos_s structures.  The nevents argument determines the size of
00121         eventlist.
00122
00123         The data_out argument provides space for extra out data provided by
00124         specific filters.  The data_available argument's contents specified the
00125         space available in the data pool on input, and contains the amount still
00126         remaining on output.  If the KEVENT_FLAG_STACK_DATA flag is specified on
00127         the system call, the data is allocated from the pool in stack order
00128         instead of typical heap order.
00129
00130         If timeout is a non-NULL pointer, it specifies a maximum interval to wait
00131         for an event, which will be interpreted as a struct timespec.  If timeout
00132         is a NULL pointer, both kevent() and kevent64() wait indefinitely.  To
00133         effect a poll, the flags argument to kevent64() or kevent_qos() can
00134         include the KEVENT_FLAG_IMMEDIATE value to indicate an immediate timeout.
00135         Alternatively, the timeout argument should be non-NULL, pointing to a
00136         zero-valued timespec structure.  The same array may be used for the
00137         changelist and eventlist.
00138
00139         The EV_SET() macro is provided for ease of initializing a kevent
00140         structure. Similarly, EV_SET64() initializes a kevent64_s structure and
00141         EV_SET_QOS() initializes a kevent_qos_s structure.
00142
00143         The kevent, kevent64_s and kevent_qos_s structures are defined as:
00144
00145         struct kevent {
00146                 uintptr_t       ident;          identifier for this event
00147                 int16_t         filter;         filter for event
00148                 uint16_t        flags;          general flags
00149                 uint32_t        fflags;         filter-specific flags
00150                 intptr_t        data;           filter-specific data
00151                 void            *udata;         opaque user data identifier
00152         };
00153
00154         struct kevent64_s {
00155                 uint64_t        ident;          identifier for this event
00156                 int16_t         filter;         filter for event
00157                 uint16_t        flags;          general flags
00158                 uint32_t        fflags;         filter-specific flags
00159                 int64_t         data;           filter-specific data
00160                 uint64_t        udata;          opaque user data identifier
00161                 uint64_t        ext[2];         filter-specific extensions
00162         };
00163
00164         struct kevent_qos_s {
00165                 uint64_t        ident;          identifier for this event
00166                 int16_t         filter;         filter for event
00167                 uint16_t        flags;          general flags
00168                 uint32_t        qos;            quality of service when servicing event
00169                 uint64_t        udata;          opaque user data identifier
00170                 uint32_t        fflags;         filter-specific flags
00171                 uint32_t        xflags;         extra filter-specific flags
00172                 int64_t         data;           filter-specific data
00173                 uint64_t        ext[4];         filter-specific extensions
00174         };
00175
00176         ----
00177
00178         The fields of struct kevent, struct kevent64_s and struct kevent_qos_s
00179         are:
00180
00181         ident       Value used to identify the source of the event.  The exact
00182                      interpretation is determined by the attached filter, but often
00183                      is a file descriptor.
00184
00185         filter      Identifies the kernel filter used to process this event.  The
00186                      pre-defined system filters are described below.
00187
00188         flags       Actions to perform on the event.
00189
00190         fflags      Filter-specific flags.
00191
00192         data        Filter-specific data value.
00193
00194         udata       Opaque user-defined value passed through the kernel unchanged.
00195                      It can optionally be part of the uniquing decision of the
00196                      kevent system
```

```
00197
00198         In addition, struct kevent64_s contains:
00199
00200         ext[2]     This field stores extensions for the event's filter. What type
00201                     of extension depends on what type of filter is being used.
00202
00203         In addition, struct kevent_qos_s contains:
00204
00205         xflags     Extra filter-specific flags.
00206
00207         ext[4]     The QoS variant provides twice as many extension values for
00208                     filter-specific uses.
00209
00210         ----
00211
00212         The flags field can contain the following values:
00213
00214         EV_ADD          Adds the event to the kqueue.  Re-adding an existing event
00215                         will modify the parameters of the original event, and not
00216                         result in a duplicate entry.  Adding an event
00217                         automatically enables it, unless overridden by the
00218                         EV_DISABLE flag.
00219
00220         EV_ENABLE       Permit kevent,() kevent64() and kevent_qos() to return the
00221                         event if it is triggered.
00222
00223         EV_DISABLE      Disable the event so kevent,() kevent64() and kevent_qos()
00224                         will not return it.  The filter itself is not disabled.
00225
00226         EV_DELETE       Removes the event from the kqueue.  Events which are
00227                         attached to file descriptors are automatically deleted on
00228                         the last close of the descriptor.
00229
00230         EV_RECEIPT      This flag is useful for making bulk changes to a kqueue
00231                         without draining any pending events. When passed as input,
00232                         it forces EV_ERROR to always be returned.  When a filter
00233                         is successfully added, the data field will be zero.
00234
00235         EV_ONESHOT      Causes the event to return only the first occurrence of
00236                         the filter being triggered.  After the user retrieves the
00237                         event from the kqueue, it is deleted.
00238
00239         EV_CLEAR        After the event is retrieved by the user, its state is
00240                         reset.  This is useful for filters which report state
00241                         transitions instead of the current state.  Note that some
00242                         filters may automatically set this flag internally.
00243
00244         EV_EOF          Filters may set this flag to indicate filter-specific EOF
00245                         condition.
00246
00247         EV_OOBAND       Read filter on socket may set this flag to indicate the
00248                         presence of out of band data on the descriptor.
00249
00250         EV_ERROR        See RETURN VALUES below.
00251
00252         ----
00253
00254         The predefined system filters are listed below.  Arguments may be passed
00255         to and from the filter via the data, fflags and optionally xflags fields
00256         in the kevent, kevent64_s or kevent_qos_s structure.
00257
00258         EVFILT_READ      Takes a file descriptor as the identifier, and returns
00259                         whenever there is data available to read.  The behavior
00260                         of the filter is slightly different depending on the
00261                         descriptor type.
00262
00263                         Sockets
00264                             Sockets which have previously been passed to
00265                             listen() return when there is an incoming connection
00266                             pending.  data contains the size of the listen
00267                             backlog.
00268
00269                             Other socket descriptors return when there is data
00270                             to be read, subject to the SO_RCVLOWAT value of the
00271                             socket buffer.  This may be overridden with a per-
00272                             filter low water mark at the time the filter is
00273                             added by setting the NOTE_LOWAT flag in fflags, and
00274                             specifying the new low water mark in data.  The
00275                             derived per filter low water mark value is, however,
00276                             bounded by socket receive buffer's high and low
00277                             water mark values.  On return, data contains the
00278                             number of bytes of protocol data available to read.
00279
00280                             The presence of EV_OOBAND in flags, indicates the
00281                             presence of out of band data on the socket data
00282                             equal to the potential number of OOB bytes availble
00283                             to read.
```

```
00284
00285                           If the read direction of the socket has shutdown,
00286                           then the filter also sets EV_EOF in flags, and
00287                           returns the socket error (if any) in fflags.  It is
00288                           possible for EOF to be returned (indicating the
00289                           connection is gone) while there is still data
00290                           pending in the socket buffer.
00291
00292                       Vnodes
00293                           Returns when the file pointer is not at the end of
00294                           file.  data contains the offset from current
00295                           position to end of file, and may be negative.
00296
00297                       Fifos, Pipes
00298                           Returns when there is data to read; data contains
00299                           the number of bytes available.
00300
00301                           When the last writer disconnects, the filter will
00302                           set EV_EOF in flags.  This may be cleared by passing
00303                           in EV_CLEAR, at which point the filter will resume
00304                           waiting for data to become available before
00305                           returning.
00306
00307                       Device nodes
00308                           Returns when there is data to read from the device;
00309                           data contains the number of bytes available.  If the
00310                           device does not support returning number of bytes,
00311                           it will not allow the filter to be attached.
00312                           However, if the NOTE_LOWAT flag is specified and the
00313                           data field contains 1 on input, those devices will
00314                           attach - but cannot be relied upon to provide an
00315                           accurate count of bytes to be read on output.
00316
00317           EVFILT_EXCEPT    Takes a descriptor as the identifier, and returns
00318                           whenever one of the specified exceptional conditions has
00319                           occurred on the descriptor. Conditions are specified in
00320                           fflags.  Currently, this filter can be used to monitor
00321                           the arrival of out-of-band data on a socket descriptor
00322                           using the filter flag NOTE_OOB.
00323
00324                           If the read direction of the socket has shutdown, then
00325                           the filter also sets EV_EOF in flags, and returns the
00326                           socket error (if any) in fflags.
00327
00328           EVFILT_WRITE     Takes a file descriptor as the identifier, and returns
00329                           whenever it is possible to write to the descriptor.  For
00330                           sockets, pipes and fifos, data will contain the amount
00331                           of space remaining in the write buffer.  The filter will
00332                           set EV_EOF when the reader disconnects, and for the fifo
00333                           case, this may be cleared by use of EV_CLEAR.  Note that
00334                           this filter is not supported for vnodes.
00335
00336                           For sockets, the low water mark and socket error
00337                           handling is identical to the EVFILT_READ case.
00338
00339           EVFILT_AIO       This filter is currently unsupported.
00340
00341           EVFILT_VNODE     Takes a file descriptor as the identifier and the events
00342                           to watch for in fflags, and returns when one or more of
00343                           the requested events occurs on the descriptor.  The
00344                           events to monitor are:
00345
00346                           NOTE_DELETE    The unlink() system call was called on
00347                                           the file referenced by the descriptor.
00348
00349                           NOTE_WRITE     A write occurred on the file referenced
00350                                           by the descriptor.
00351
00352                           NOTE_EXTEND    The file referenced by the descriptor was
00353                                           extended.
00354
00355                           NOTE_ATTRIB    The file referenced by the descriptor had
00356                                           its attributes changed.
00357
00358                           NOTE_LINK      The link count on the file changed.
00359
00360                           NOTE_RENAME    The file referenced by the descriptor was
00361                                           renamed.
00362
00363                           NOTE_REVOKE    Access to the file was revoked via
00364                                           revoke(2) or the underlying fileystem was
00365                                           unmounted.
00366
00367                           NOTE_FUNLOCK   The file was unlocked by calling flock(2)
00368                                           or close(2)
00369
00370                           NOTE_LEASE_DOWNGRADE
```

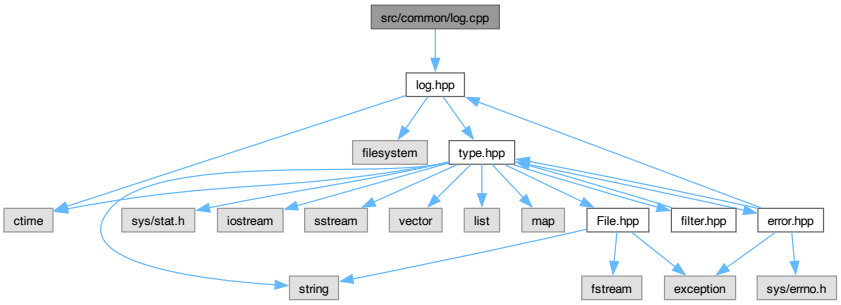
```
00371                                   A lease break to downgrade the lease to
00372                                   read lease is requested on the file
00373                                   referenced by the descriptor.
00374
00375                           NOTE_LEASE_RELEASE
00376                                   A lease break to release the lease is
00377                                   requested on the file or directory
00378                                   referenced by the descriptor.
00379
00380                           On return, fflags contains the filter-specific flags
00381                           which are associated with the triggered events seen by
00382                           this filter.
00383
00384                   EVFILT_PROC      Takes the process ID to monitor as the identifier and
00385                           the events to watch for in fflags, and returns when the
00386                           process performs one or more of the requested events.
00387                           If a process can normally see another process, it can
00388                           attach an event to it.  The events to monitor are:
00389
00390                           NOTE_EXIT    The process has exited.
00391
00392                           NOTE_EXITSTATUS
00393                                       The process has exited and its exit status
00394                                       is in filter specific data. Valid only on
00395                                       child processes and to be used along with
00396                                       NOTE_EXIT.
00397
00398                           NOTE_FORK    The process created a child process via
00399                                       fork(2) or similar call.
00400
00401                           NOTE_EXEC    The process executed a new process via
00402                                       execve(2) or similar call.
00403
00404                           NOTE_SIGNAL  The process was sent a signal. Status can
00405                                       be checked via waitpid(2) or similar call.
00406
00407                           NOTE_REAP    The process was reaped by the parent via
00408                                       wait(2) or similar call. Deprecated, use
00409                                       NOTE_EXIT.
00410
00411                           On return, fflags contains the events which triggered
00412                           the filter.
00413
00414                   EVFILT_SIGNAL    Takes the signal number to monitor as the identifier and
00415                           returns when the given signal is generated for the
00416                           process.  This coexists with the signal() and
00417                           sigaction() facilities, and has a lower precedence.
00418                           Only signals sent to the process, not to a particular
00419                           thread, will trigger the filter. The filter will record
00420                           all attempts to deliver a signal to a process, even if
00421                           the signal has been marked as SIG_IGN.  Event
00422                           notification happens before normal signal delivery
00423                           processing.  data returns the number of times the signal
00424                           has been generated since the last call to kevent().
00425                           This filter automatically sets the EV_CLEAR flag
00426                           internally.
00427
00428                   EVFILT_MACHPORT  Takes the name of a mach port, or port set, in ident and
00429                           waits until a message is enqueued on the port or port
00430                           set. When a message is detected, but not directly
00431                           received by the kevent call, the name of the specific
00432                           port where the message is enqueued is returned in data.
00433                           If fflags contains MACH_RCV_MSG, the ext[0] and ext[1]
00434                           flags are assumed to contain a pointer to the buffer
00435                           where the message is to be received and the size of the
00436                           receive buffer, respectively.  If MACH_RCV_MSG is
00437                           specifed, yet the buffer size in ext[1] is zero, The
00438                           space for the buffer may be carved out of the data_out
00439                           area provided to kevent_qos() if there is enough space
00440                           remaining there.
00441
00442                   EVFILT_TIMER     Establishes an interval timer identified by ident where
00443                           data specifies the timeout period (in milliseconds).
00444
00445                           fflags can include one of the following flags to specify
00446                           a different unit:
00447
00448                           NOTE_SECONDS   data is in seconds
00449
00450                           NOTE_USECONDS  data is in microseconds
00451
00452                           NOTE_NSECONDS  data is in nanoseconds
00453
00454                           NOTE_MACHTIME  data is in Mach absolute time units
00455
00456                           fflags can also include NOTE_ABSOLUTE, which establishes
00457                           an EV_ONESHOT timer with an absolute deadline instead of
```

```
00458                       an interval.  The absolute deadline is expressed in
00459                       terms of gettimeofday(2).  With NOTE_MACHTIME, the
00460                       deadline is expressed in terms of mach_absolute_time().
00461
00462                       The timer can be coalesced with other timers to save
00463                       power. The following flags can be set in fflags to
00464                       modify this behavior:
00465
00466                       NOTE_CRITICAL    override default power-saving
00467                                       techniques to more strictly respect the
00468                                       leeway value
00469
00470                       NOTE_BACKGROUND  apply more power-saving techniques to
00471                                       coalesce this timer with other timers
00472
00473                       NOTE_LEEWAY      ext[1] holds user-supplied slop in
00474                                       deadline for timer coalescing.
00475
00476                       The timer will be periodic unless EV_ONESHOT is
00477                       specified.  On return, data contains the number of times
00478                       the timeout has expired since the last arming or last
00479                       delivery of the timer event.
00480
00481                       This filter automatically sets the EV_CLEAR flag.
00482
00483           ----
00484
00485           In the ext[2] field of the kevent64_s struture, ext[0] is only used with
00486           the EVFILT_MACHPORT filter.  With other filters, ext[0] is passed through
00487           kevent64() much like udata.  ext[1] can always be used like udata.  For
00488           the use of ext[0], see the EVFILT_MACHPORT filter above.
00489
00490       RETURN VALUES
00491           The kqueue() system call creates a new kernel event queue and returns a
00492           file descriptor.  If there was an error creating the kernel event queue,
00493           a value of -1 is returned and errno set.
00494
00495           The kevent(), kevent64() and kevent_qos() system calls return the number
00496           of events placed in the eventlist, up to the value given by nevents.  If
00497           an error occurs while processing an element of the changelist and there
00498           is enough room in the eventlist, then the event will be placed in the
00499           eventlist with EV_ERROR set in flags and the system error in data.
00500           Otherwise, -1 will be returned, and errno will be set to indicate the
00501           error condition.  If the time limit expires, then kevent(), kevent64()
00502           and kevent_qos() return 0.
00503 */
```

7.14 src/common/log.cpp File Reference

```
#include "log.hpp"
```

Include dependency graph for log.cpp:

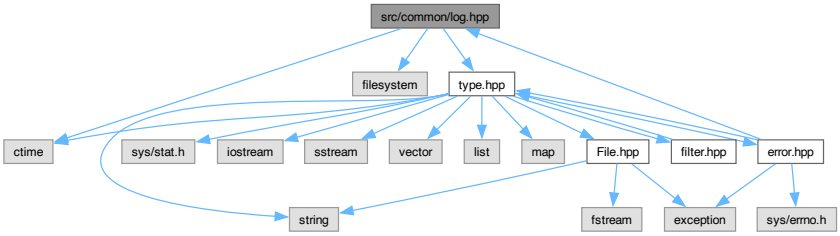


7.15 src/common/log.hpp File Reference

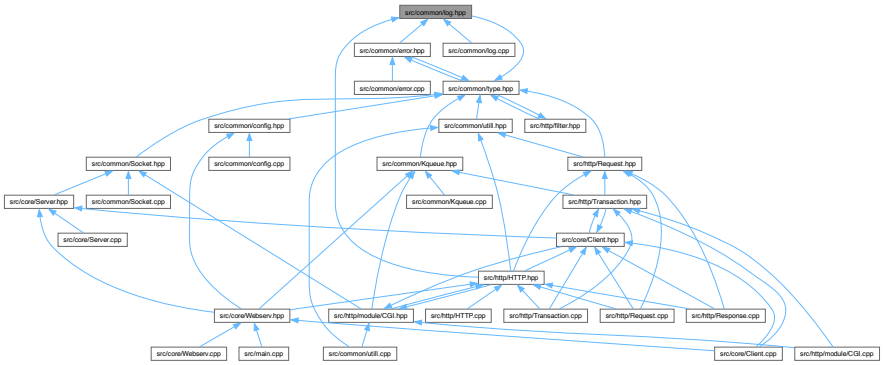
```
#include <ctime>
#include <filesystem>
```

```
#include "type.hpp"
```

Include dependency graph for log.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace log

Functions

- std::string log::logFname (void)
- std::string log::strTime (void)
- void log::timestamp (void)
- void log::print (const str_t &)
- void log::printVec (vec_str_t &, const str_t)

Variables

- const std::time_t log::begin = std::time(NULL)
- File log::history

7.16 log.hpp

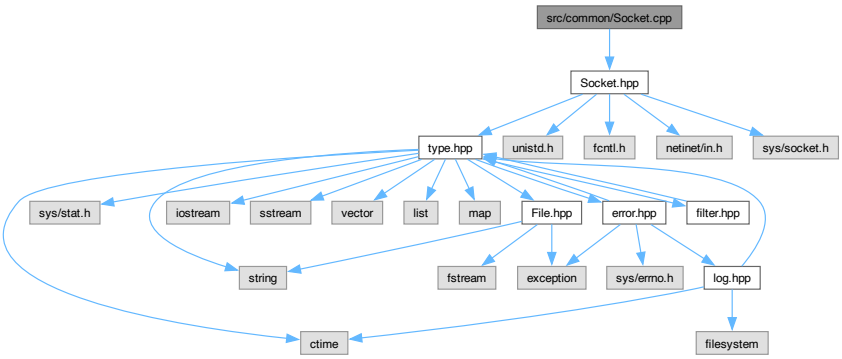
Go to the documentation of this file.

```
00001 #ifndef LOG_HPP
00002 # define LOG_HPP
00003
00004 # include <ctime>
00005 # include <filesystem>
00006
00007 # include "type.hpp"
00008
00009 namespace log {
00010     extern const std::time_t    begin;
00011     extern File                 history;
00012
00013     std::string logFname( void );
00014     std::string strTime( void );
00015
00016     void        timestamp( void );
00017     void        print( const str_t& );
00018
00019     void        printVec( vec_str_t&, const str_t );
00020 }
00021
00022 #endif
```

7.17 src/common/Socket.cpp File Reference

```
#include "Socket.hpp"
```

Include dependency graph for Socket.cpp:

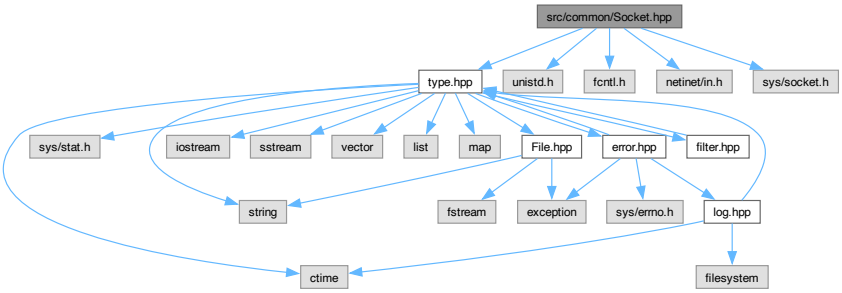


7.18 src/common/Socket.hpp File Reference

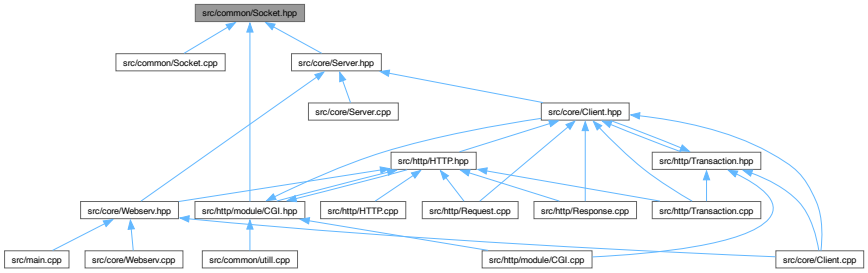
```
#include "type.hpp"
#include <unistd.h>
#include <fcntl.h>
#include <netinet/in.h>
```

```
#include <sys/socket.h>
```

Include dependency graph for Socket.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Socket

Typedefs

- typedef int fd_t
- typedef struct sockaddr sockaddr_t
- typedef struct sockaddr_in sockaddr_in_t

7.18.1 Typedef Documentation

7.18.1.1 fd_t

```
typedef int fd_t
```

7.18.1.2 sockaddr_in_t

```
typedef struct sockaddr_in sockaddr_in_t
```

7.18.1.3 sockaddr_t

```
typedef struct sockaddr sockaddr_t
```

7.19 Socket.hpp

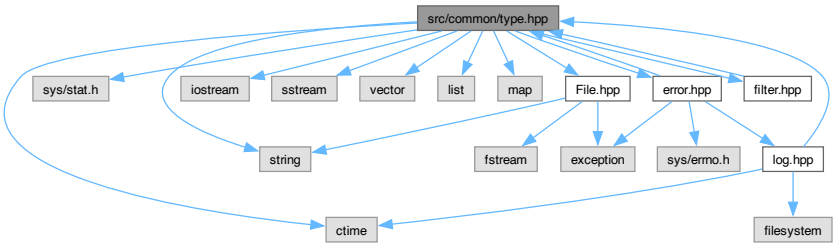
Go to the documentation of this file.

```
00001 #ifndef SOCKET_HPP
00002 # define SOCKET_HPP
00003
00004 # include "type.hpp"
00005
00006 # include <unistd.h>
00007 # include <fcntl.h>
00008
00009 # include <netinet/in.h>
00010 # include <sys/socket.h>
00011
00012 typedef int fd_t;
00013
00014 typedef struct sockaddr     sockaddr_t;
00015 typedef struct sockaddr_in  sockaddr_in_t;
00016
00017 class Socket {
00018     public:
00019         Socket();
00020         Socket(const fd_t&);
00021         Socket(const Socket&&) noexcept;
00022         virtual ~Socket();
00023
00024         Socket&           operator=(const Socket&&) noexcept;
00025
00026         sockaddr_in_t  addr;
00027         socklen_t       addr_len;
00028
00029         const fd_t&     sock() const;
00030         void            setNonblock() const;
00031
00032     private:
00033         fd_t            _sock;
00034 };
00035
00036 #endif
```

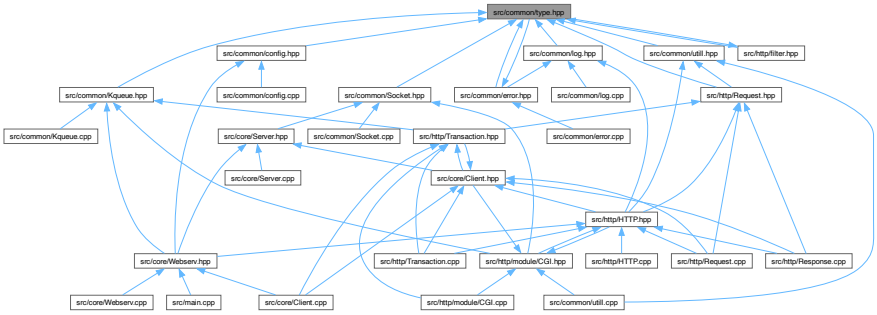
7.20 src/common/type.hpp File Reference

```
#include <string>
#include <sys/stat.h>
#include <ctime>
#include <iostream>
#include <sstream>
#include <vector>
#include <list>
#include <map>
#include "File.hpp"
#include "error.hpp"
#include "filter.hpp"
```

Include dependency graph for type.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct process_s

Macros

- #define FALSE 0
- #define TRUE 1

Typedefs

- typedef int socket_t
- typedef int port_t
- typedef int pipe_t
- typedef int stat_t
- typedef unsigned int uint_t
- typedef unsigned int bits_t
- typedef std::string str_t
- typedef str_t name_t
- typedef str_t path_t
- typedef str_t type_t
- typedef struct stat fstat_t
- typedef std::time_t ctime_t

- typedef std::stringstream sstream_t
- typedef std::ostringstream osstream_t
- typedef std::istringstream isstream_t
- template<typename T>
  using vec = std::vector<T>
- typedef std::vector< uint_t > vec_uint_t
- typedef std::vector< str_t > vec_str_t
- typedef vec_str_t::iterator vec_str_iter_t
- typedef vec_str_t vec_name_t
- typedef std::vector< char * > vec_cstr_t
- template<typename T>
  using list = std::list<T>
- template<typename Tkey, typename Tval>
  using map = std::map<Tkey, Tval>
- template<typename Tkey, typename Tval>
  using pair = std::pair<Tkey, Tval>
- typedef std::map< uint_t, str_t > map_uint_str_t
- typedef std::map< str_t, type_t > map_str_type_t
- typedef std::map< str_t, path_t > map_str_path_t
- typedef struct process_s process_t

7.20.1 Macro Definition Documentation

7.20.1.1 FALSE

```
#define FALSE 0
```

7.20.1.2 TRUE

```
#define TRUE 1
```

7.20.2 Typedef Documentation

7.20.2.1 bits_t

```
typedef unsigned int bits_t
```

7.20.2.2 ctime_t

```
typedef std::time_t ctime_t
```

7.20.2.3 fstat_t

```
typedef struct stat fstat_t
```

7.20.2.4 isstream_t

```
typedef std::istringstream isstream_t
```

7.20.2.5 list

```
template<typename T>
using list = std::list<T>
```

7.20.2.6 map

```
template<typename Tkey, typename Tval>
using map = std::map<Tkey, Tval>
```

7.20.2.7 map_str_path_t

```
typedef std::map<str_t, path_t> map_str_path_t
```

7.20.2.8 map_str_type_t

```
typedef std::map<str_t, type_t> map_str_type_t
```

7.20.2.9 map_uint_str_t

```
typedef std::map<uint_t, str_t> map_uint_str_t
```

7.20.2.10 name_t

```
typedef str_t name_t
```

7.20.2.11 osstream_t

```
typedef std::ostringstream osstream_t
```

7.20.2.12 pair

```
template<typename Tkey, typename Tval>
using pair = std::pair<Tkey, Tval>
```

7.20.2.13 path_t

```
typedef str_t path_t
```

7.20.2.14 pipe_t

```
typedef int pipe_t
```

7.20.2.15 port_t

```
typedef int port_t
```

7.20.2.16 process_t

```
typedef struct process_s process_t
```

7.20.2.17 socket_t

```
typedef int socket_t
```

7.20.2.18 sstream_t

```
typedef std::stringstream sstream_t
```

7.20.2.19 stat_t

```
typedef int stat_t
```

7.20.2.20 str_t

```
typedef std::string str_t
```

7.20.2.21 type_t

```
typedef str_t type_t
```

7.20.2.22 uint_t

```
typedef unsigned int uint_t
```

7.20.2.23 vec

```
template<typename T>
using vec = std::vector<T>
```

7.20.2.24 vec_cstr_t

```
typedef std::vector<char*> vec_cstr_t
```

7.20.2.25 vec_name_t

```
typedef vec_str_t vec_name_t
```

7.20.2.26 vec_str_iter_t

```
typedef vec_str_t::iterator vec_str_iter_t
```

7.20.2.27 vec_str_t

```
typedef std::vector<str_t> vec_str_t
```

7.20.2.28 vec_uint_t

```
typedef std::vector<uint_t> vec_uint_t
```

7.21 type.hpp

Go to the documentation of this file.

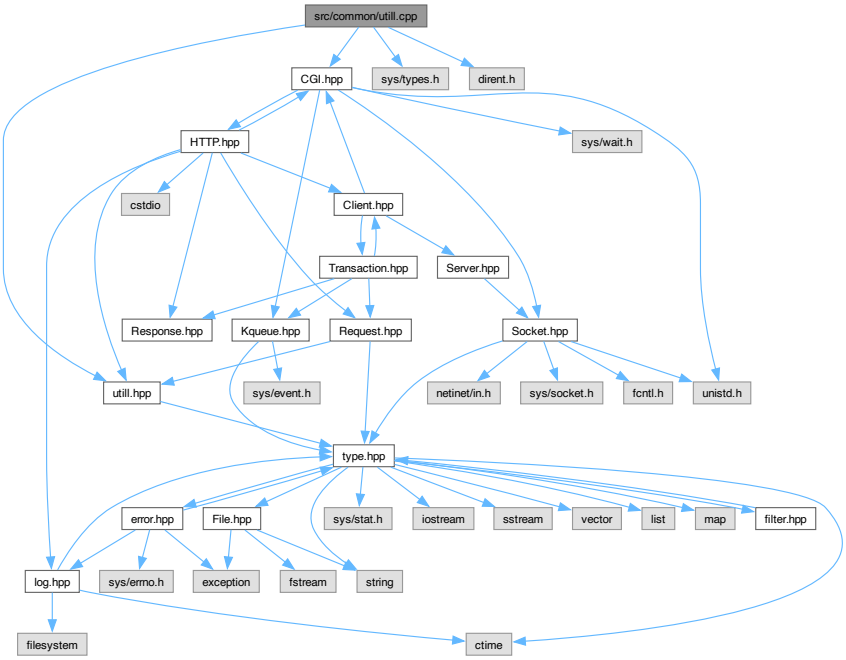
```
00001 #ifndef TYPE_HPP
00002 # define TYPE_HPP
00003
00004 # include <string>
00005
00006 # define FALSE          0
00007 # define TRUE           1
00008
00009 typedef int                           socket_t;
00010 typedef int                           port_t;
00011 typedef int                           pipe_t;
00012 typedef int                           stat_t;
00013
00014 typedef unsigned int                  uint_t;
00015 typedef unsigned int                  bits_t;
00016
00017 typedef std::string                   str_t;
00018
00019 /*
00020     name_t: the name of file, means when use it, should be combined with path
00021     path_t: absolute/relative path
00022     type_t: the type of mime
00023 */
00024
00025 typedef str_t                        name_t;
00026 typedef str_t                        path_t;
00027 typedef str_t                        type_t;
00028
00029 /* POSIX */
00030 # include <sys/stat.h>
00031 # include <ctime>
00032
00033 typedef struct stat                   fstat_t;
00034 typedef std::time_t                   ctime_t;
00035
00036 /* I/O */
00037 # include <iostream>
```

```
00038 # include <sstream>
00039
00040 typedef std::stringstream           sstream_t;
00041 typedef std::ostringstream          osstream_t;
00042 typedef std::istringstream          isstream_t;
00043
00044 /* Container */
00045 # include <vector>
00046 template <typename T>
00047 using vec = std::vector<T>;
00048
00049 typedef std::vector<uint_t>         vec_uint_t;
00050 typedef std::vector<str_t>          vec_str_t;
00051 typedef vec_str_t::iterator         vec_str_iter_t;
00052 typedef vec_str_t                  vec_name_t;
00053 typedef std::vector<char*>          vec_cstr_t;
00054
00055 # include <list>
00056 template <typename T>
00057 using list = std::list<T>;
00058
00059 # include <map>
00060 template <typename Tkey, typename Tval>
00061 using map = std::map<Tkey, Tval>;
00062
00063 template <typename Tkey, typename Tval>
00064 using pair = std::pair<Tkey, Tval>;
00065
00066 typedef std::map<uint_t, str_t>     map_uint_str_t;
00067 typedef std::map<str_t, type_t>     map_str_type_t;
00068 typedef std::map<str_t, path_t>     map_str_path_t;
00069
00070 /* STRUCT */
00071 typedef struct process_s {
00072     process_s( void );
00073
00074     void        reset( void );
00075
00076     pid_t       pid;
00077     stat_t        stat;
00078     pipe_t        fd[2];
00079
00080     vec_str_t  argv;
00081     vec_str_t  env;
00082 }   process_t;
00083
00084 # include "File.hpp"
00085
00086 # include "error.hpp"
00087 # include "filter.hpp"
00088
00089 #endif
00090
```

7.22 src/common/utill.cpp File Reference

```
#include "utill.hpp"
#include "CGI.hpp"
#include <sys/types.h>
#include <dirent.h>
```

Include dependency graph for utill.cpp:



Functions

- bool dead (const process_t &procs)
- bool found (const size_t &pos)
- str_t token (isstream_t &iss, const char &delim)
- bool getInfo (const str_t &target, fstat_t &info)
- bool isExist (const str_t &target)
- bool isDir (const fstat_t &info)
- ctime_t getNow (void)
- str_t timeToStr (const ctime_t &time)
- void errpageScript (sstream_t &page, const uint_t &status, const str_t &explanation)
- void autoindexScript (const path_t &dir_path, sstream_t &body)

7.22.1 Function Documentation

7.22.1.1 autoindexScript()

```
void autoindexScript (
            const path_t & dir_path,
            sstream_t & body)
```

Here is the call graph for this function:

Here is the caller graph for this function:

7.22.1.2 dead()

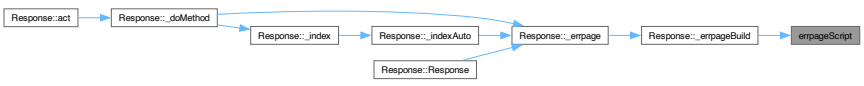
```
bool dead (
            const process_t & procs)
```

Here is the caller graph for this function:

7.22.1.3 errpageScript()

```
void errpageScript (
            sstream_t & page,
            const uint_t & status,
            const str_t & explanation)
```

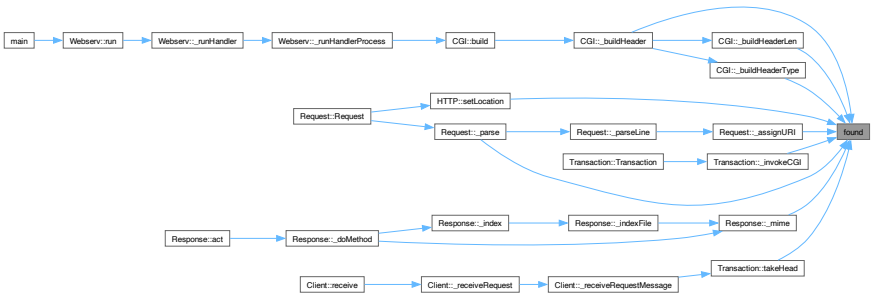
Here is the caller graph for this function:



7.22.1.4 found()

```
bool found (
            const size_t & pos)
```

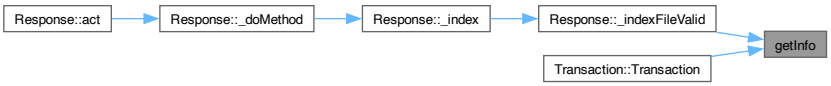
Here is the caller graph for this function:



7.22.1.5 getInfo()

```
bool getInfo (
            const str_t & target,
            fstat_t & info)
```

Here is the caller graph for this function:



7.22.1.6 getNow()

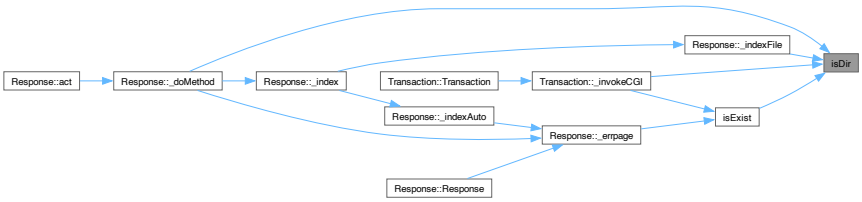
```
ctime_t getNow (
            void )
```

Here is the caller graph for this function:

7.22.1.7 isDir()

```
bool isDir (
            const fstat_t & info)
```

Here is the caller graph for this function:

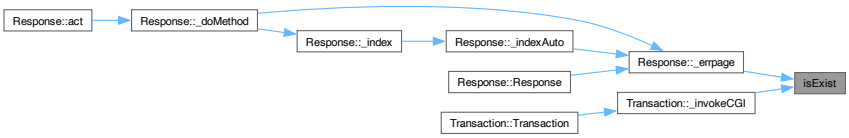


7.22.1.8 isExist()

```
bool isExist (
            const str_t & target)
```

Here is the call graph for this function:

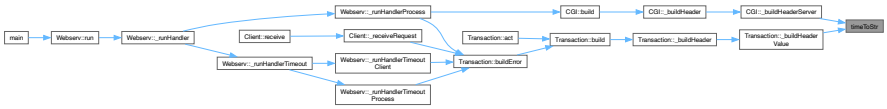
Here is the caller graph for this function:



7.22.1.9 timeToStr()

```
str_t timeToStr (
            const ctime_t & time)
```

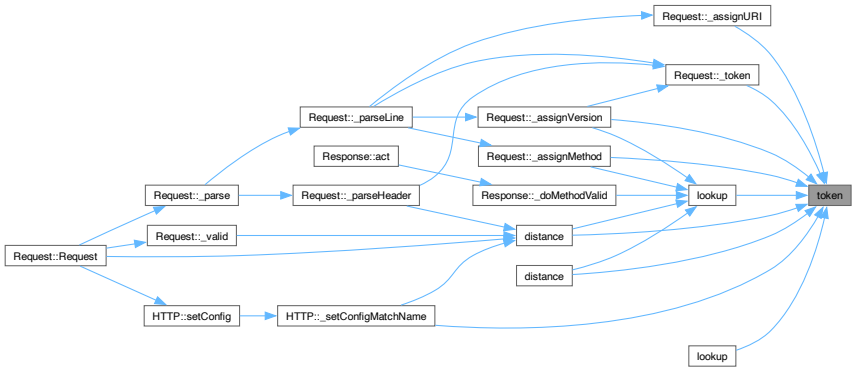
Here is the caller graph for this function:



7.22.1.10 token()

```
str_t token (
            isstream_t & iss,
            const char & delim)
```

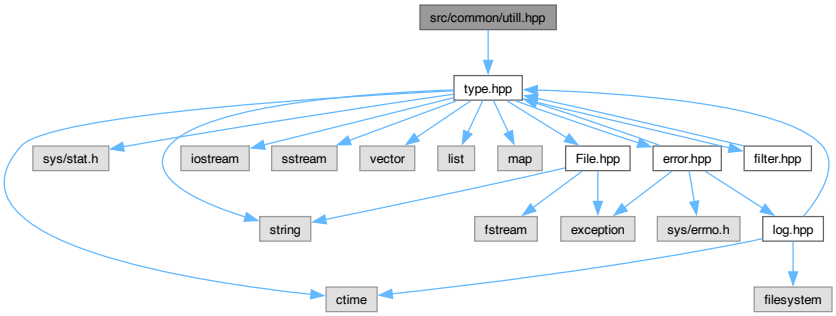
Here is the caller graph for this function:



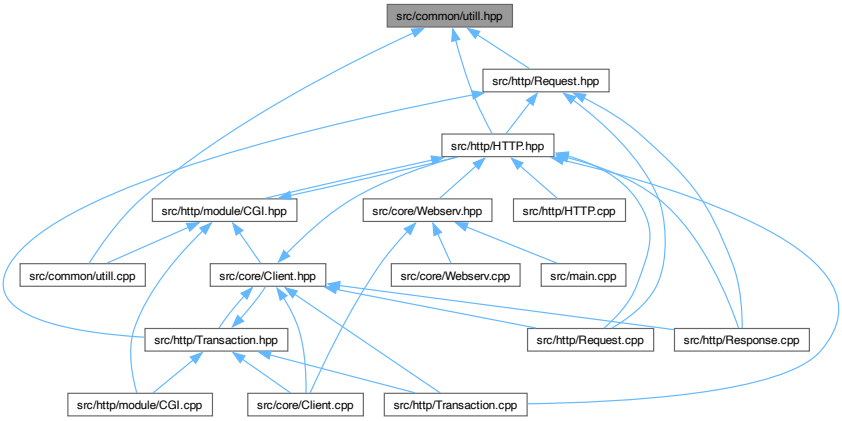
7.23 src/common/utill.hpp File Reference

```
#include "type.hpp"
```

Include dependency graph for utill.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define NOT_FOUND -1

Typedefs

- typedef std::streamsize streamsize_t
- typedef std::streampos streampos_t

Functions

- bool dead (const process_t &)
- bool found (const size_t &)
- str_t token (isstream_t &, const char &)
- bool getInfo (const str_t &, fstat_t &)
- bool isExist (const str_t &)
- bool isDir (const fstat_t &)
- ctime_t getNow (void)
- str_t timeToStr (const ctime_t &)
- void errpageScript (sstream_t &, const uint_t &, const str_t &)
- void autoindexScript (const path_t &, sstream_t &)
- template<typename Container, typename Target>
  Container::iterator lookup (Container &obj, Target token)
- template<typename Container, typename Target>
  Container::const_iterator lookup (const Container &obj, Target token)
- template<typename Container, typename Target>
  ssize_t distance (Container &obj, Target token)
- template<typename Container, typename Target>
  ssize_t distance (const Container &obj, const Target token)
- template<typename Stream>
  streamsize_t streamsize (Stream &obj)

7.23.1 Macro Definition Documentation

7.23.1.1 NOT_FOUND

```
#define NOT_FOUND -1
```

7.23.2 Typedef Documentation

7.23.2.1 streampos_t

```
typedef std::streampos streampos_t
```

7.23.2.2 streamsize_t

```
typedef std::streamsize streamsize_t
```

7.23.3 Function Documentation

7.23.3.1 autoindexScript()

```
void autoindexScript (
            const path_t & dir_path,
            sstream_t & body)
```

Here is the call graph for this function:

Here is the caller graph for this function:

7.23.3.2 dead()

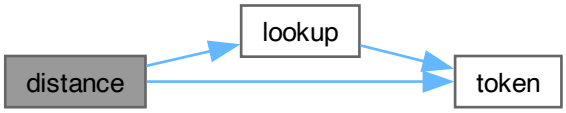
```
bool dead (
            const process_t & procs)
```

Here is the caller graph for this function:

7.23.3.3 distance() [1/2]

```
template<typename Container, typename Target>
ssize_t distance (
            const Container & obj,
            const Target token)
```

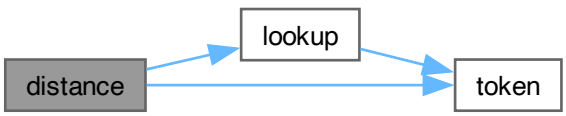
Here is the call graph for this function:



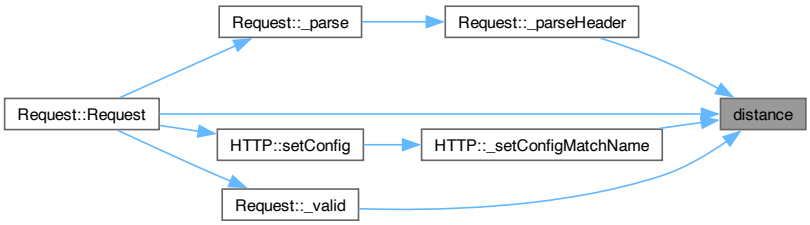
7.23.3.4 distance() [2/2]

```
template<typename Container, typename Target>
ssize_t distance (
            Container & obj,
            Target token)
```

Here is the call graph for this function:



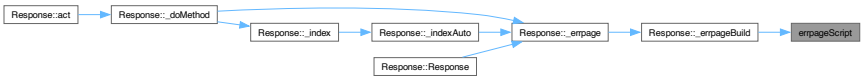
Here is the caller graph for this function:



7.23.3.5 errpageScript()

```
void errpageScript (
            sstream_t & page,
            const uint_t & status,
            const str_t & explanation)
```

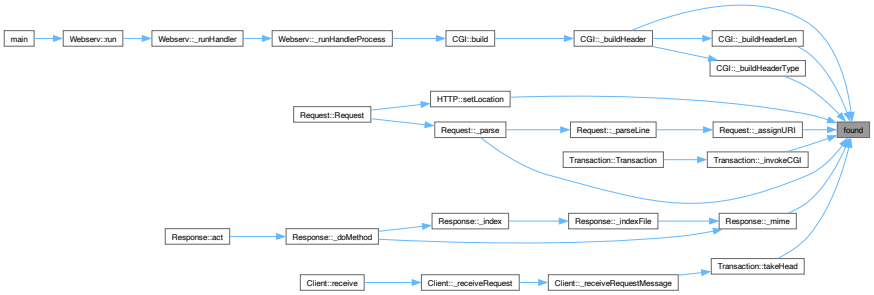
Here is the caller graph for this function:



7.23.3.6 found()

```
bool found (
            const size_t & pos)
```

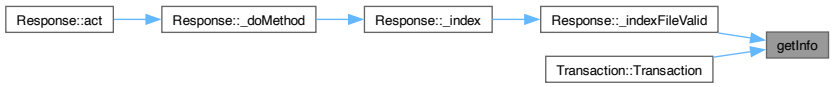
Here is the caller graph for this function:



7.23.3.7 getInfo()

```
bool getInfo (
            const str_t & target,
            fstat_t & info)
```

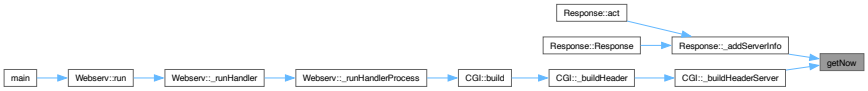
Here is the caller graph for this function:



7.23.3.8 getNow()

```
ctime_t getNow (
            void )
```

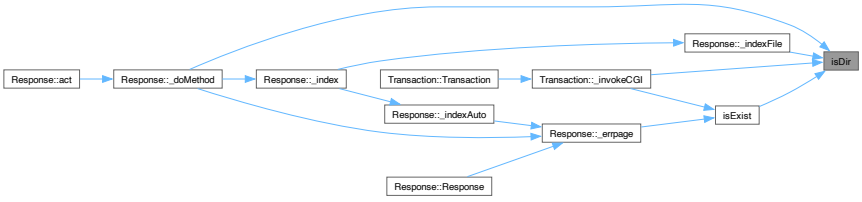
Here is the caller graph for this function:



7.23.3.9 isDir()

```
bool isDir (
            const fstat_t & info)
```

Here is the caller graph for this function:

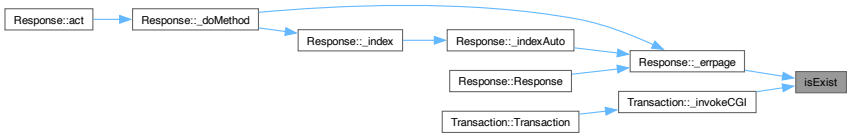


7.23.3.10 isExist()

```
bool isExist (
            const str_t & target)
```

Here is the call graph for this function:

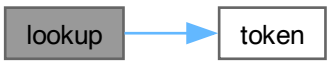
Here is the caller graph for this function:



7.23.3.11 lookup() [1/2]

```
template<typename Container, typename Target>
Container::const_iterator lookup (
            const Container & obj,
            Target token)
```

Here is the call graph for this function:

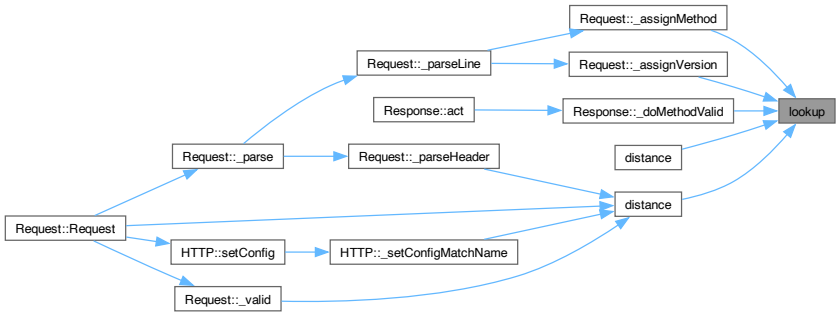


7.23.3.12 lookup() [2/2]

```
template<typename Container, typename Target>
Container::iterator lookup (
            Container & obj,
            Target token)
```

Here is the call graph for this function:

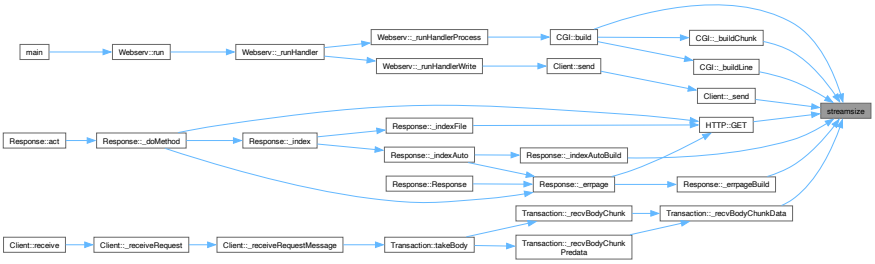
Here is the caller graph for this function:



7.23.3.13 streamsize()

```
template<typename Stream>
streamsize_t streamsize (
            Stream & obj)
```

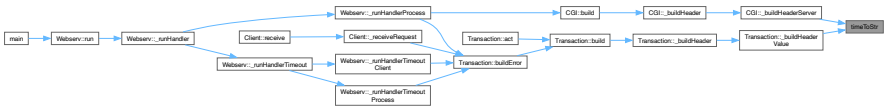
Here is the caller graph for this function:



7.23.3.14 timeToStr()

```
str_t timeToStr (
            const ctime_t & time)
```

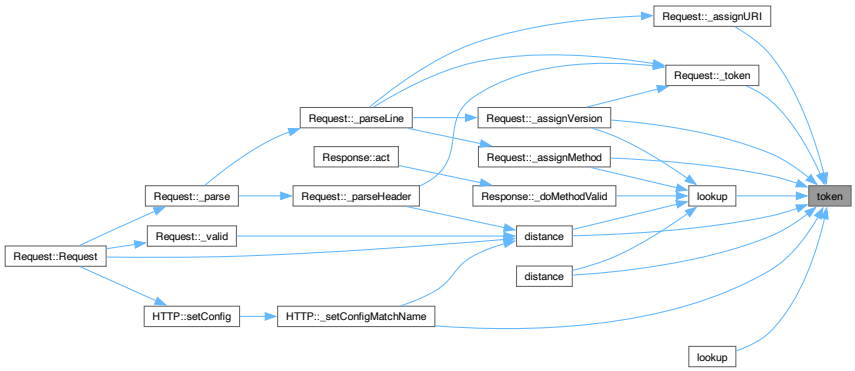
Here is the caller graph for this function:



7.23.3.15 token()

```
str_t token (
            isstream_t & iss,
            const char & delim)
```

Here is the caller graph for this function:



7.24 utill.hpp

Go to the documentation of this file.

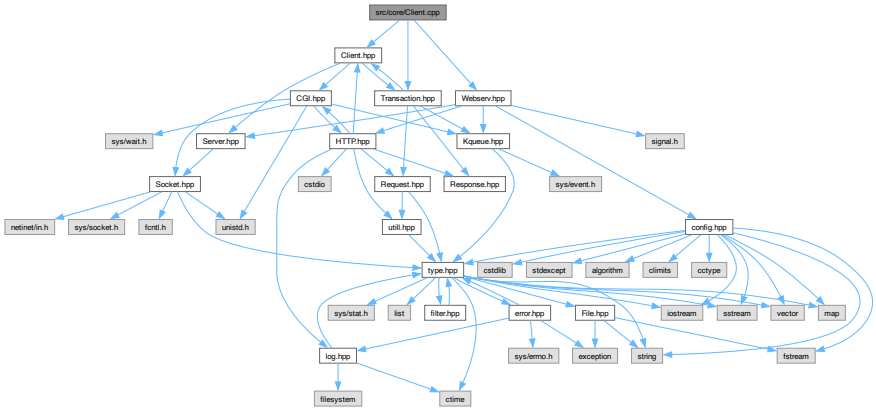
```
00001 #ifndef UTILL_HPP
00002 # define UTILL_HPP
00003
00004 # include "type.hpp"
00005
00006 # define NOT_FOUND -1
00007
00008 /* UTILL */
00009 bool            dead( const process_t& );
00010 bool            found( const size_t& );
00011 str_t          token( isstream_t&, const char& );
00012
00013 /* FILE INFO */
00014 bool            getInfo( const str_t&, fstat_t& );
00015 bool            isExist( const str_t& );
00016 bool            isDir( const fstat_t& );
00017
```

```
00018 /* TIME */
00019 ctime_t          getNow( void );
00020 str_t          timeToStr( const ctime_t& );
00021
00022 /* BUILT-IN SCRIPT */
00023 void            errpageScript( sstream_t&, const uint_t&, const str_t& );
00024 void            autoindexScript( const path_t&, sstream_t& );
00025
00026 /* TEMPLETE FUNCTION */
00027 /* lookup: Get iterator for matching with the token whitin container object */
00028 template<typename Container, typename Target> typename Container::iterator
00029 lookup( Container& obj, Target token ) { return std::find( obj.begin(), obj.end(), token ); }
00030
00031 template<typename Container, typename Target> typename Container::const_iterator
00032 lookup( const Container& obj, Target token ) { return std::find( obj.begin(), obj.end(), token ); }
00033
00034 /* distance: Get index of container object matching with the token */
00035 template<typename Container, typename Target> ssize_t
00036 distance( Container& obj, Target token ) {
00037     typename Container::iterator    iter = lookup( obj, token );
00038
00039     if ( iter != obj.end() ) return std::distance( obj.begin(), iter );
00040     else return -1;
00041 }
00042
00043 template<typename Container, typename Target> ssize_t
00044 distance( const Container& obj, const Target token ) {
00045     typename Container::const_iterator  iter = lookup( obj, token );
00046
00047     if ( iter != obj.end() ) return std::distance( obj.begin(), iter );
00048     else return -1;
00049 }
00050
00051 /* streamsize: Get size of stream object
00052 not actually alloc'd size, but from the current positon to end */
00053 typedef std::streamsize streamsize_t;
00054 typedef std::streampos  streampos_t;
00055
00056 template<typename Stream> streamsize_t
00057 streamsize( Stream& obj ) {
00058     streampos_t curnt = obj.tellg();
00059     if ( curnt == streampos_t( ERROR ) ) return 0;
00060
00061     obj.seekg( 0, std::ios::end );
00062     streampos_t end = obj.tellg();
00063     if ( end == streampos_t( ERROR ) ) return 0;
00064
00065     streampos_t result = end - curnt;
00066     obj.seekg( curnt );
00067
00068     return result;
00069 }
00070
00071 #endif
```

7.25 src/core/Client.cpp File Reference

```
#include "Client.hpp"
#include "Webserv.hpp"
#include "Transaction.hpp"
```

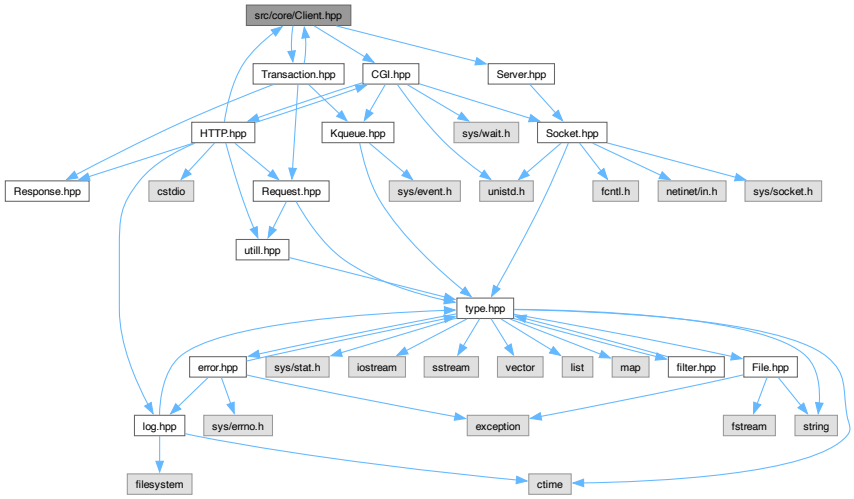
Include dependency graph for Client.cpp:



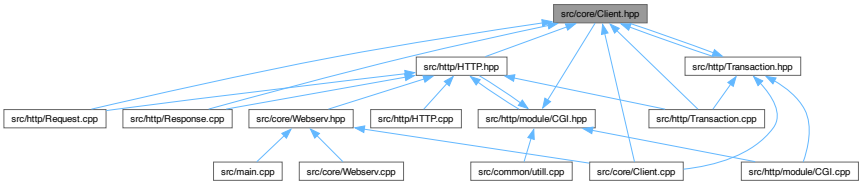
7.26 src/core/Client.hpp File Reference

```
#include "CGI.hpp"
#include "Server.hpp"
#include "Transaction.hpp"
```

Include dependency graph for Client.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Client

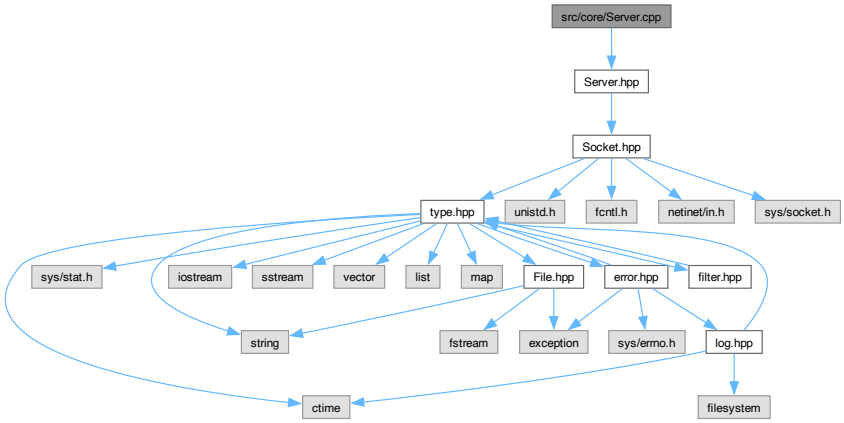
7.27 Client.hpp

Go to the documentation of this file.

```
00001 #ifndef CLIENT_HPP
00002 # define CLIENT_HPP
00003
00004 # include "CGI.hpp"
00005 # include "Server.hpp"
00006 # include "Transaction.hpp"
00007
00008 class Kqueue;
00009
00010 class Client: public Socket {
00011     public:
00012         Client() = delete;
00013         Client(const Server&);
00014         Client(Client&&) noexcept;
00015         ~Client();
00016
00017         Client&           operator=(Client&&) noexcept;
00018         bool             operator==(const Client&) const; /* For searching map container sock_Client */
00019
00020         const Server& server() const;
00021
00022         message_t      in;
00023         message_t      out;
00024
00025         Transaction* trans;
00026         process_t      subproc;
00027
00028         bool            receive(Kqueue&);
00029         bool            send();
00030         void            reset();
00031
00032     private:
00033         const Server& _srv;
00034         char            _buff[SIZE_BUFF_RECV];
00035
00036         void            _receiveRequest(Kqueue&, ssize_t&);
00037         bool            _receiveRequestMessage(Kqueue&, ssize_t&);
00038         void            _receiveRequestDo(Kqueue&);
00039
00040         ssize_t         _send(sstream_t&);
00041 };
00042
00043 #endif
```

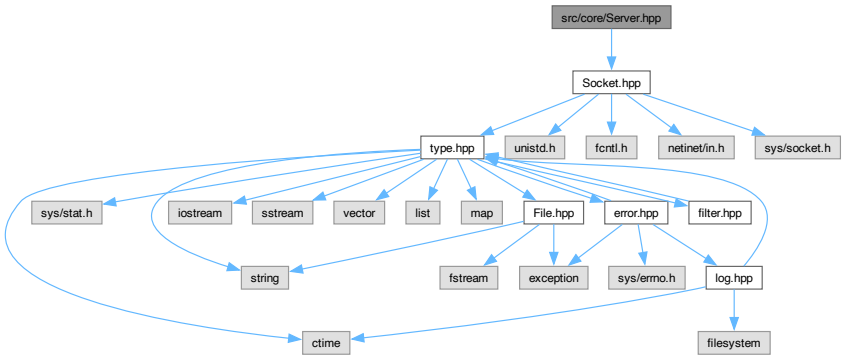
7.28 src/core/Server.cpp File Reference

`#include "Server.hpp"`
Include dependency graph for Server.cpp:

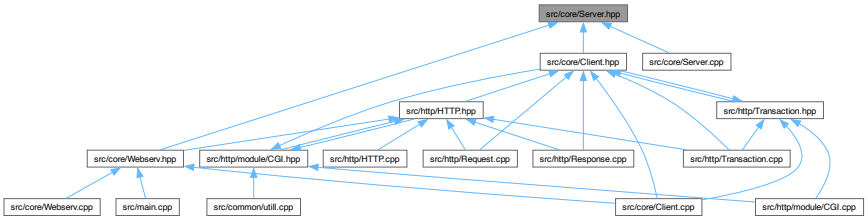


7.29 src/core/Server.hpp File Reference

`#include "Socket.hpp"`
Include dependency graph for Server.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Server

Macros

- #define MAX_CLIENT 32
- #define DEFAULT 0

7.29.1 Macro Definition Documentation

7.29.1.1 DEFAULT

```
#define DEFAULT 0
```

7.29.1.2 MAX_CLIENT

```
#define MAX_CLIENT 32
```

7.30 Server.hpp

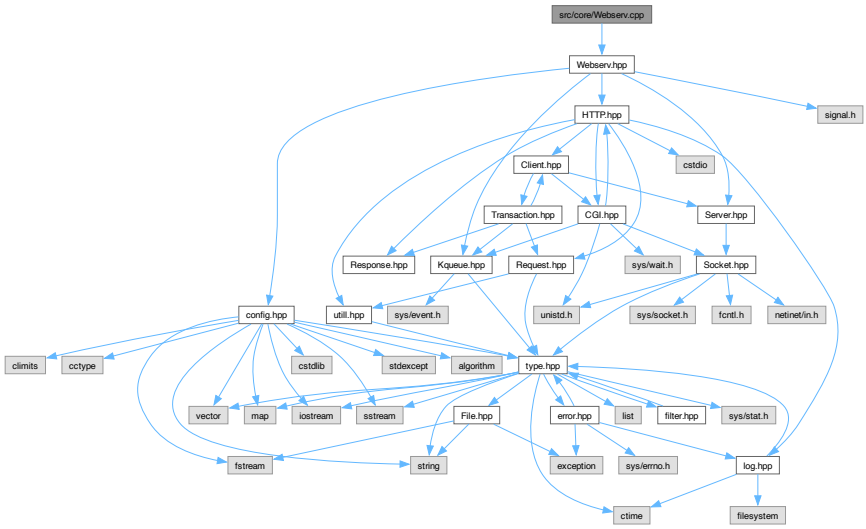
Go to the documentation of this file.

```
00001 #ifndef SERVER_HPP
00002 # define SERVER_HPP
00003
00004 # include "Socket.hpp"
00005
00006 # define MAX_CLIENT 32
00007 # define DEFAULT    0
00008
00009 class Server: public Socket {
00010     public:
00011         Server(const port_t&);
00012         Server(Server&&) noexcept;
00013         ~Server();
00014
00015         Server&                   operator=(Server&&) noexcept;
00016
00017         const vec<config_t>&   config() const;
00018
00019         void                    configAdd(const config_t&);
00020
00021     private:
00022         vec<config_t>          _conf;
00023
00024         void                    _open(const port_t&);
00025         void                    _openSetAddr(const int&);
00026 };
00027
00028
00029 #endif
```

7.31 src/core/Webserv.cpp File Reference

```
#include "Webserv.hpp"
```

Include dependency graph for Webserv.cpp:



Variables

- int udata [4]

7.31.1 Detailed Description

Author

Donghyun Kim (6moguai9@gmail.com)

Version

0.1

Date

2025-01-11

Copyright

Copyright (c) 2025

7.31.2 Variable Documentation

7.31.2.1 udata

```
int udata[4]
```

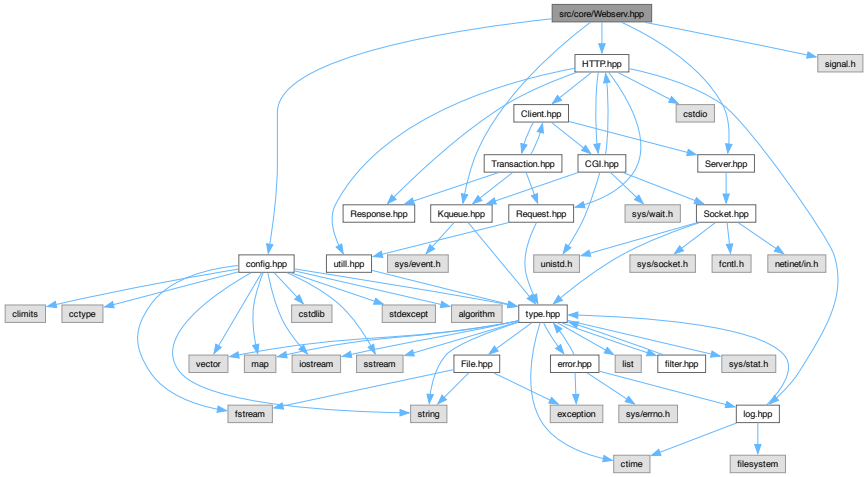
Initial value:

```
= {
    READ_SERVER,
    READ_CLIENT,
    TIMER_CLIENT_IDLE,
    TIMER_CLIENT_RQST
}
```

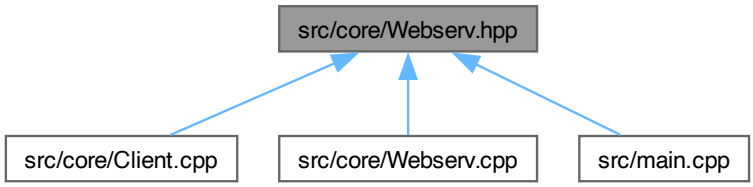
7.32 src/core/Webserv.hpp File Reference

```
#include "config.hpp"
#include "Kqueue.hpp"
#include "Server.hpp"
#include "HTTP.hpp"
#include <signal.h>
```

Include dependency graph for Webserv.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Webserv
- struct Webserv::list_s
- struct Webserv::map_s

Macros

- #define WEBSERV_HPP
- #define TIMEOUT_CLIENT_IDLE 30000
- #define TIMEOUT_CLIENT_RQST 30000
- #define TIMEOUT_PROCS 60000

Enumerations

- enum state_e { SUSPEND , RUNNING }
- enum udata_e { READ_SERVER , READ_CLIENT , TIMER_CLIENT_IDLE , TIMER_CLIENT_RQST }

Variables

- int udata [4]

7.32.1 Detailed Description

Author

Donghyun Kim (6moguai9@gmail.com)

Version

0.1

Date

2025-01-11

Copyright

7.32.2 Macro Definition Documentation

7.32.2.1 TIMEOUT_CLIENT_IDLE

```
#define TIMEOUT_CLIENT_IDLE 30000
```

7.32.2.2 TIMEOUT_CLIENT_RQST

```
#define TIMEOUT_CLIENT_RQST 30000
```

7.32.2.3 TIMEOUT_PROCS

```
#define TIMEOUT_PROCS 60000
```

7.32.2.4 WEBSERV_HPP

```
#define WEBSERV_HPP
```

7.32.3 Enumeration Type Documentation

7.32.3.1 state_e

```
enum state_e
```

Enumerator

SUSPEND	
RUNNING	

7.32.3.2 udata_e

```
enum udata_e
```

Enumerator

READ_SERVER	
READ_CLIENT	
TIMER_CLIENT_IDLE	
TIMER_CLIENT_RQST	

7.32.4 Variable Documentation

7.32.4.1 udata

```
int udata[4]  [extern]
```

7.33 Webserv.hpp

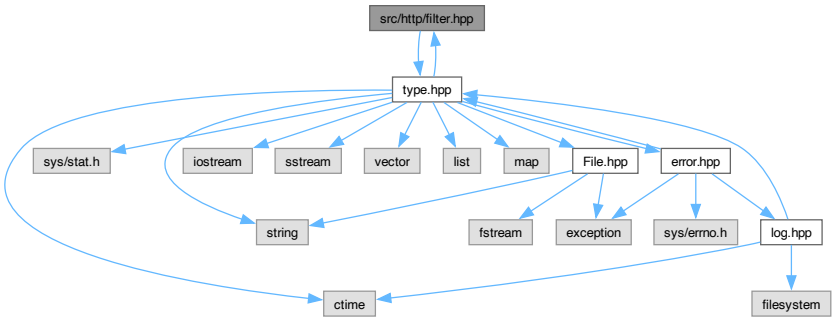
Go to the documentation of this file.

```
00001
00011
00012 # define WEBSERV_HPP
00013
00014 # include "config.hpp"
00015 # include "Kqueue.hpp"
00016 # include "Server.hpp"
00017 # include "HTTP.hpp"
00018
00019 # include <signal.h>
00020
00021 /* Timeout in ms */
00022 # define TIMEOUT_CLIENT_IDLE 30000
00023 # define TIMEOUT_CLIENT_RQST 30000
00024 # define TIMEOUT_PROCS 60000
00025
00026 enum state_e {
00027     SUSPEND,
00028     RUNNING
00029 };
00030
00031 enum udata_e {
00032     READ_SERVER,
00033     READ_CLIENT,
00034     TIMER_CLIENT_IDLE,
00035     TIMER_CLIENT_RQST
00036 };
00037
00038 extern int udata[4];
00039
00040 class Webserv {
00041     public:
00042         Webserv() = delete;
00043         Webserv(Kqueue&);
00044         ~Webserv();
00045
00046         int     state;
00047
00048         void    init(const char*);
00049         void    run();
00050
00051     private:
00052         Kqueue&                  _evnt;
00053
00054         struct list_s {
00055             list<Server>        srv;
00056             list<Client>        cl;
00057         }                       _list;
00058
00059         struct map_s {
00060             map<port_t, fd_t>  port_sock;
00061             map<fd_t, Server&>    sock_srv;
00062             map<fd_t, Client&>    sock_cl;
00063         }                       _map;
00064
00065         void    _loadConfig(const char*, vec<config_t>&);
00066         void    _loadConfigPrint(const vec<config_t>&) const;
00067         void    _initServer(vec<config_t>&);
00068         void    _initServerCreate(const port_t&);
00069         void    _initScheme();
00070
00071         void    _runHandler();
00072         bool    _runHandlerDisconnect(const event_t&);
00073         void    _runHandlerRead(const event_t&);
00074         void    _runHandlerReadServer(const uintptr_t&);
00075         void    _runHandlerReadClient(const event_t&);
00076         void    _runHandlerWrite(const event_t&);
00077         void    _runHandlerProcess(const event_t&);
00078         void    _runHandlerTimeout(const event_t&);
00079         void    _runHandlerTimeoutClient(const event_t&, Client&);
00080         void    _runHandlerTimeoutProcess(const event_t&);
00081
00082         void    _disconnect(const event_t&);
00083         void    _disconnectPrint(const event_t&);
00084 };
00085
00086 #endif
```

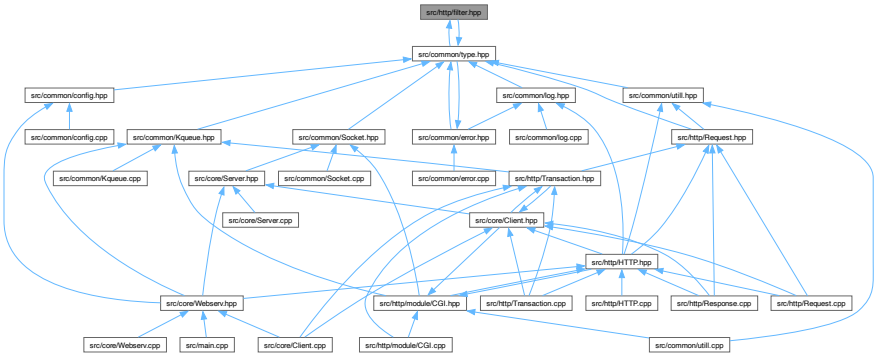
7.34 src/http/filter.hpp File Reference

```
#include "type.hpp"
```

Include dependency graph for filter.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct keys_t
- struct http_s
- struct location_s
- struct config_s
- struct request_line_t
- struct request_header_s
- struct response_line_s
- struct response_header_s

Macros

- #define NONE 0
- #define CR '\r'
- #define LF '\n'
- #define SP ' '

- #define CRLF "\r\n"
- #define SIZE_CRLF 2
- #define MSG_END "\r\n\r\n"
- #define SIZE_MSG_END 4
- #define DEFAULT 0

Typedefs

- typedef std::map< method_e, bool > map_method_bool_t
- typedef struct http_s http_t
- typedef struct location_s location_t
- typedef std::vector< location_t > vec_location_t
- typedef struct config_s config_t
- typedef std::vector< config_t > vec_config_t
- typedef struct request_header_s request_header_t
- typedef struct response_line_s response_line_t
- typedef struct response_header_s response_header_t

Enumerations

- enum method_e {
  GET , POST , DELETE , NOT_ALLOWED ,
  UNKNOWN }
- enum version_e {
  VERSION_9 , VERSION_10 , VERSION_11 , VERSION_20 ,
  NOT_SUPPORTED }
- enum connection_e { CN_KEEP_ALIVE , CN_CLOSE , CN_UNKNOWN }
- enum transfer_enc_e { TE_IDENTITY , TE_CHUNKED , TE_UNKNOWN }
- enum header_in_e {
  IN_HOST , IN_CONNECTION , IN_TRANSFER_ENC , IN_CONTENT_LEN ,
  IN_CONTENT_TYPE , IN_COOKIE }
- enum header_out_e {
  OUT_SERVER , OUT_DATE , OUT_CONNECTION , OUT_TRANSFER_ENC ,
  OUT_CONTENT_LEN , OUT_CONTENT_TYPE , OUT_LOCATION , OUT_ALLOW ,
  OUT_SET_COOKIE }
- enum cgi_env_e {
  SERVER_NAME , SERVER_PORT , SERVER_PROTOCOL , REMOTE_ADDR ,
  REMOTE_HOST , GATEWAY_INTERFACE , REQUEST_METHOD , SCRIPT_NAME ,
  CONTENT_LENGTH , CONTENT_TYPE , PATH_INFO , PATH_TRANSLATED ,
  QUERY_STRING , UPLOAD_DIR , HTTP_COOKIE }

Variables

- const str_t str_method [ ]
- const str_t str_version [ ]
- const str_t str_connection [ ]
- const str_t str_transfer_enc [ ]

7.34.1 Macro Definition Documentation

7.34.1.1 CR

```
#define CR '\r'
```

7.34.1.2 CRLF

```
#define CRLF "\r\n"
```

7.34.1.3 DEFAULT

```
#define DEFAULT 0
```

7.34.1.4 LF

```
#define LF '\n'
```

7.34.1.5 MSG_END

```
#define MSG_END "\r\n\r\n"
```

7.34.1.6 NONE

```
#define NONE 0
```

7.34.1.7 SIZE_CRLF

```
#define SIZE_CRLF 2
```

7.34.1.8 SIZE_MSG_END

```
#define SIZE_MSG_END 4
```

7.34.1.9 SP

```
#define SP ' '
```

7.34.2 Typedef Documentation

7.34.2.1 config_t

```
typedef struct config_s config_t
```

7.34.2.2 http_t

```
typedef struct http_s http_t
```

7.34.2.3 location_t

```
typedef struct location_s location_t
```

7.34.2.4 map_method_bool_t

```
typedef std::map<method_e, bool> map_method_bool_t
```

7.34.2.5 request_header_t

```
typedef struct request_header_s request_header_t
```

7.34.2.6 response_header_t

```
typedef struct response_header_s response_header_t
```

7.34.2.7 response_line_t

```
typedef struct response_line_s response_line_t
```

7.34.2.8 vec_config_t

```
typedef std::vector<config_t> vec_config_t
```

7.34.2.9 vec_location_t

```
typedef std::vector<location_t> vec_location_t
```

7.34.3 Enumeration Type Documentation

7.34.3.1 cgi_env_e

```
enum cgi_env_e
```

Enumerator

SERVER_NAME	
SERVER_PORT	
SERVER_PROTOCOL	
REMOTE_ADDR	
REMOTE_HOST	
GATEWAY_INTERFACE	
REQUEST_METHOD	
SCRIPT_NAME	
CONTENT_LENGTH	
CONTENT_TYPE	
PATH_INFO	
PATH_TRANSLATED	
QUERY_STRING	
UPLOAD_DIR	
HTTP_COOKIE	

7.34.3.2 connection_e

enum connection_e

Enumerator

CN_KEEP_ALIVE	
CN_CLOSE	
CN_UNKNOWN	

7.34.3.3 header_in_e

enum header_in_e

Enumerator

IN_HOST	
IN_CONNECTION	
IN_TRANSFER_ENC	
IN_CONTENT_LEN	
IN_CONTENT_TYPE	
IN_COOKIE	

7.34.3.4 header_out_e

enum header_out_e

Enumerator

OUT_SERVER	
OUT_DATE	
OUT_CONNECTION	
OUT_TRANSFER_ENC	
OUT_CONTENT_LEN	
OUT_CONTENT_TYPE	
OUT_LOCATION	
OUT_ALLOW	
OUT_SET_COOKIE	

7.34.3.5 method_e

enum method_e

Enumerator

GET	
POST	
DELETE	
NOT_ALLOWED	
UNKNOWN	

7.34.3.6 transfer_enc_e

```
enum transfer_enc_e
```

Enumerator

TE_IDENTITY	
TE_CHUNKED	
TE_UNKNOWN	

7.34.3.7 version_e

```
enum version_e
```

Enumerator

VERSION_9	
VERSION_10	
VERSION_11	
VERSION_20	
NOT_SUPPORTED	

7.34.4 Variable Documentation

7.34.4.1 str_connection

```
const str_t str_connection[]
```

Initial value:

```
= {
    "keep-alive",
    "close"
}
```

7.34.4.2 str_method

```
const str_t str_method[]
```

Initial value:

```
= {
    "GET",
    "POST",
    "DELETE"
}
```

7.34.4.3 str_transfer_enc

```
const str_t str_transfer_enc[]
```

Initial value:

```
= {
    "identity",
    "chunked"
}
```

7.34.4.4 str_version

```
const str_t str_version[]
```

Initial value:

```
= {
    "0.9",
    "1.0",
    "1.1",
    "2.0"
}
```

7.35 filter.hpp

Go to the documentation of this file.

```
00001 #ifndef FILTER_HPP
00002 # define FILTER_HPP
00003
00004 # include "type.hpp"
00005
00006 /*
00007     token   = 1*tchar
00008     tchar   = "!" / "#" / "$" / "%" / "&" / "'" / "*"
00009                   / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
00010                   / DIGIT / ALPHA; any VCHAR, except delimiters
00011
00012     OWS     = *( SP / HTAB ); optional whitespace
00013     RWS     = 1*( SP / HTAB ); required whitespace
00014     BWS     = OWS; "bad" whitespace
00015
00016     See iana.org for method, header and status list
00017     method: https://www.iana.org/assignments/http-methods/http-methods.xhtml
00018     header: https://www.iana.org/assignments/http-fields/http-fields.xhtml
00019             also https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
00020     status:
00021     https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml
00022 */
00023
00024 # define NONE           0
00025
00026 # define CR             '\r'
00027 # define LF             '\n'
00028 # define SP             ' '
00029
00030 # define CRLF           "\r\n"
00031 # define SIZE_CRLF      2
00032
00033 # define MSG_END        "\r\n\r\n"
00034 # define SIZE_MSG_END   4
00035
00036 # define DEFAULT        0
00037
00038 /* STRINGS - for assigning vector values (because of the version
00039 limitation, not available to use the vector argument initialize list) */
00040 const str_t str_method[] = {
00041     "GET",
00042     "POST",
00043     "DELETE"
00044 };
00045
00046 const str_t str_version[] = {
00047     "0.9",
00048     "1.0",
00049     "1.1",
00050     "2.0"
00051 };
00052
00053 const str_t str_connection[] = {
00054     "keep-alive",
00055     "close"
00056 };
00057
00058 const str_t str_transfer_enc[] = {
00059     "identity",
00060     "chunked"
00061 };
00062
```

```
00063 /* ENUM */
00064 enum method_e {
00065     GET,
00066     POST,
00067     DELETE,
00068     NOT_ALLOWED,
00069     UNKNOWN
00070 };
00071
00072 enum version_e {
00073     VERSION_9,
00074     VERSION_10,
00075     VERSION_11,
00076     VERSION_20,
00077     NOT_SUPPORTED
00078 };
00079
00080 enum connection_e {
00081     CN_KEEP_ALIVE,
00082     CN_CLOSE,
00083     CN_UNKNOWN
00084 };
00085
00086 enum transfer_enc_e {
00087     TE_IDENTITY,
00088     TE_CHUNKED,
00089     TE_UNKNOWN
00090 };
00091
00092 enum header_in_e {
00093     IN_HOST,
00094     IN_CONNECTION,
00095     IN_TRANSFER_ENC,
00096     IN_CONTENT_LEN,
00097     IN_CONTENT_TYPE,
00098     IN_COOKIE
00099 };
00100
00101 enum header_out_e {
00102     OUT_SERVER,
00103     OUT_DATE,
00104     OUT_CONNECTION,
00105     OUT_TRANSFER_ENC,
00106     OUT_CONTENT_LEN,
00107     OUT_CONTENT_TYPE,
00108     OUT_LOCATION,
00109     OUT_ALLOW,
00110     OUT_SET_COOKIE
00111 };
00112
00113 enum cgi_env_e {
00114     SERVER_NAME,
00115     SERVER_PORT,
00116     SERVER_PROTOCOL,
00117     REMOTE_ADDR,
00118     REMOTE_HOST,
00119     GATEWAY_INTERFACE,
00120     REQUEST_METHOD,
00121     SCRIPT_NAME,
00122     CONTENT_LENGTH,
00123     CONTENT_TYPE,
00124     PATH_INFO,
00125     PATH_TRANSLATED,
00126     QUERY_STRING,
00127     UPLOAD_DIR,
00128     HTTP_COOKIE
00129 };
00130
00131 /* STRUCT - Key, Http: core values for implementing HTTP */
00132 typedef struct {
00133     vec_str_t          header_in;
00134     vec_str_t          header_out;
00135     map_uint_str_t        status;
00136     map_str_type_t        mime;
00137 }   keys_t;
00138
00139 typedef std::map<method_e, bool> map_method_bool_t;
00140
00141 typedef struct http_s {
00142     str_t              signature;
00143     vec_str_t          version;
00144     vec_str_t          method;
00145
00146     type_t                type_unknown;
00147     vec_str_t          encoding;
00148     vec_str_t          connection;
00149 }   http_t;
```

```
00150
00151 /* STRUCT - Configs (Server, Location) */
00152 struct config_s;
00153
00154 typedef struct location_s {
00155     location_s( const config_s& );
00156
00157     str_t               path;
00158     path_t                 root;
00159     path_t                 rewrite;
00160
00161     vec_uint_t             allow;
00162
00163     bool                cgi;
00164     path_t                 upload;
00165
00166     vec_name_t             index;
00167     bool                index_auto;
00168 }   location_t;
00169
00170 typedef std::vector<location_t> vec_location_t;
00171
00172 typedef struct config_s {
00173     config_s( void );
00174
00175     vec_str_t           names;
00176     port_t                 listen;
00177     path_t                 root;
00178
00179     name_t                 file_40x;
00180     name_t                 file_50x;
00181
00182     size_t              client_max_body;
00183
00184     vec_location_t         locations;
00185 }   config_t;
00186
00187 typedef std::vector<config_t> vec_config_t;
00188
00189 /* STRUCT - Request, Response */
00190 typedef struct {
00191     version_e           version;
00192     method_e             method;
00193     path_t                 uri;
00194     str_t               query;
00195
00196 }   request_line_t;
00197
00198 typedef struct request_header_s {
00199     request_header_s( void );
00200
00201     str_t               host;
00202     // date_t                 date;
00203     connection_e         connection;
00204     transfer_enc_e         transfer_encoding;
00205     size_t               content_length;
00206     type_t                 content_type;
00207
00208     str_t               cookie;
00209
00210     vec_uint_t             list;
00211
00212 }   request_header_t;
00213
00214 typedef struct response_line_s {
00215     response_line_s( void );
00216
00217     unsigned             version: 2;
00218     uint_t                 status;
00219
00220 }   response_line_t;
00221
00222 typedef struct response_header_s {
00223     response_header_s( void );
00224
00225     str_t               server;
00226     ctime_t                date;
00227     connection_e         connection;
00228     transfer_enc_e         transfer_encoding;
00229     size_t               content_length;
00230     type_t                 content_type;
00231     str_t               location;
00232     vec_uint_t             allow;
00233
00234     str_t               cookie;
00235
00236     vec_uint_t             list;
```

```
00237
00238 }   response_header_t;
00239
00240 #endif
00241
00242 /*
00243 5.1. Field Names
00244 A field name labels the corresponding field value as having the semantics
00245 defined by that name. For example, the Date header field is defined in
00246 Section 6.6.1 as containing the origination timestamp for the message in which
00247 it appears.
00248
00249   field-name     = token
00250 Field names are case-insensitive and ought to be registered within the
00251 "Hypertext Transfer Protocol (HTTP) Field Name Registry"; see Section 16.3.1.
00252
00253 The interpretation of a field does not change between minor versions of the same
00254 major HTTP version, though the default behavior of a recipient in the absence of
00255 such a field can change. Unless specified otherwise, fields are defined for all
00256 versions of HTTP. In particular, the Host and Connection fields ought to be
00257 recognized by all HTTP implementations whether or not they advertise conformance
00258 with HTTP/1.1.
00259
00260 New fields can be introduced without changing the protocol version if their
00261 defined semantics allow them to be safely ignored by recipients that do not
00262 recognize them; see Section 16.3.
00263
00264 A proxy MUST forward unrecognized header fields unless the field name is listed
00265 in the Connection header field (Section 7.6.1) or the proxy is specifically
00266 configured to block, or otherwise transform, such fields. Other recipients
00267 SHOULD ignore unrecognized header and trailer fields. Adhering to these
00268 requirements allows HTTP's functionality to be extended without updating or
00269 removing deployed intermediaries.
00270
00271 5.2. Field Lines and Combined Field Value
00272 Field sections are composed of any number of field lines, each with a field name
00273 (see Section 5.1) identifying the field, and a field line value that conveys
00274 data for that instance of the field.
00275
00276 When a field name is only present once in a section, the combined field value
00277 for that field consists of the corresponding field line value. When a field name
00278 is repeated within a section, its combined field value consists of the list of
00279 corresponding field line values within that section, concatenated in order, with
00280 each field line value separated by a comma.
00281
00282 For example, this section:
00283
00284 Example-Field: Foo, Bar
00285 Example-Field: Baz
00286 contains two field lines, both with the field name "Example-Field". The first
00287 field line has a field line value of "Foo, Bar", while the second field line
00288 value is "Baz". The field value for "Example-Field" is the list "Foo, Bar, Baz".
00289
00290 5.3. Field Order
00291 A recipient MAY combine multiple field lines within a field section that have
00292 the same field name into one field line, without changing the semantics of the
00293 message, by appending each subsequent field line value to the initial field line
00294 value in order, separated by a comma (",") and optional whitespace (OWS, defined
00295 in Section 5.6.3). For consistency, use comma SP.
00296
00297 The order in which field lines with the same name are received is therefore
00298 significant to the interpretation of the field value; a proxy MUST NOT change
00299 the order of these field line values when forwarding a message.
00300
00301 This means that, aside from the well-known exception noted below, a sender MUST
00302 NOT generate multiple field lines with the same name in a message (whether in
00303 the headers or trailers) or append a field line when a field line of the same
00304 name already exists in the message, unless that field's definition allows
00305 multiple field line values to be recombined as a comma-separated list (i.e., at
00306 least one alternative of the field's definition allows a comma-separated list,
00307 such as an ABNF rule of #(values) defined in Section 5.6.1).
00308
00309 Note: In practice, the "Set-Cookie" header field ([COOKIE]) often appears in a
00310 response message across multiple field lines and does not use the list syntax,
00311 violating the above requirements on multiple field lines with the same field
00312 name. Since it cannot be combined into a single field value, recipients ought to
00313 handle "Set-Cookie" as a special case while processing fields. (See Appendix
00314 A.2.3 of [Kri2001] for details.)
00315
00316 The order in which field lines with differing field names are received in a
00317 section is not significant. However, it is good practice to send header fields
00318 that contain additional control data first, such as Host on requests and Date on
00319 responses, so that implementations can decide when not to handle a message as
00320 early as possible.
00321
00322 A server MUST NOT apply a request to the target resource until it receives the
00323 entire request header section, since later header field lines might include
```

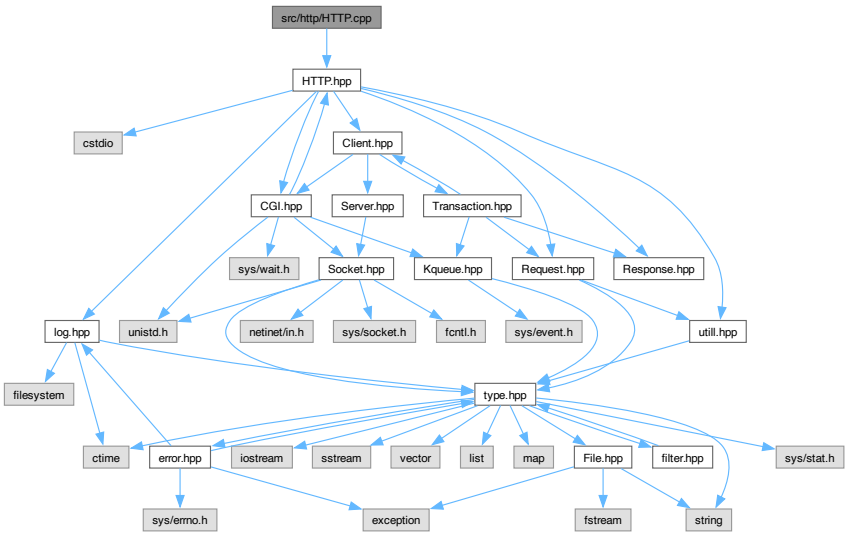
```
00324 conditionals, authentication credentials, or deliberately misleading duplicate
00325 header fields that could impact request processing.
00326
00327 5.4. Field Limits
00328 HTTP does not place a predefined limit on the length of each field line, field
00329 value, or on the length of a header or trailer section as a whole, as described
00330 in Section 2. Various ad hoc limitations on individual lengths are found in
00331 practice, often depending on the specific field's semantics.
00332
00333 A server that receives a request header field line, field value, or set of
00334 fields larger than it wishes to process MUST respond with an appropriate 4xx
00335 (Client Error) status code. Ignoring such header fields would increase the
00336 server's vulnerability to request smuggling attacks (Section 11.2 of
00337 [HTTP/1.1]).
00338
00339 A client MAY discard or truncate received field lines that are larger than the
00340 client wishes to process if the field semantics are such that the dropped
00341 value(s) can be safely ignored without changing the message framing or response
00342 semantics.
00343
00344 5.5. Field Values
00345 HTTP field values consist of a sequence of characters in a format defined by the
00346 field's grammar. Each field's grammar is usually defined using ABNF ([RFC5234]).
00347
00348   field-value    = *field-content
00349   field-content  = field-vchar
00350                    [ 1*( SP / HTAB / field-vchar ) field-vchar ]
00351   field-vchar    = VCHAR / obs-text
00352   obs-text       = %x80-FF
00353 A field value does not include leading or trailing whitespace. When a specific
00354 version of HTTP allows such whitespace to appear in a message, a field parsing
00355 implementation MUST exclude such whitespace prior to evaluating the field value.
00356
00357 Field values are usually constrained to the range of US-ASCII characters
00358 [USASCII]. Fields needing a greater range of characters can use an encoding,
00359 such as the one defined in [RFC8187]. Historically, HTTP allowed field content
00360 with text in the ISO-8859-1 charset [ISO-8859-1], supporting other charsets only
00361 through use of [RFC2047] encoding. Specifications for newly defined fields
00362 SHOULD limit their values to visible US-ASCII octets (VCHAR), SP, and HTAB. A
00363 recipient SHOULD treat other allowed octets in field content (i.e., obs-text) as
00364 opaque data.
00365
00366 Field values containing CR, LF, or NUL characters are invalid and dangerous, due
00367 to the varying ways that implementations might parse and interpret those
00368 characters; a recipient of CR, LF, or NUL within a field value MUST either
00369 reject the message or replace each of those characters with SP before further
00370 processing or forwarding of that message. Field values containing other CTL
00371 characters are also invalid; however, recipients MAY retain such characters for
00372 the sake of robustness when they appear within a safe context (e.g., an
00373 application-specific quoted string that will not be processed by any downstream
00374 HTTP parser).
00375
00376 Fields that only anticipate a single member as the field value are referred to
00377 as singleton fields.
00378
00379 Fields that allow multiple members as the field value are referred to as
00380 list-based fields. The list operator extension of Section 5.6.1 is used as a
00381 common notation for defining field values that can contain multiple members.
00382
00383 Because commas (",") are used as the delimiter between members, they need to be
00384 treated with care if they are allowed as data within a member. This is true for
00385 both list-based and singleton fields, since a singleton field might be
00386 erroneously sent with multiple members and detecting such errors improves
00387 interoperability. Fields that expect to contain a comma within a member, such as
00388 within an HTTP-date or URI-reference element, ought to be defined with
00389 delimiters around that element to distinguish any comma within that data from
00390 potential list separators.
00391
00392 For example, a textual date and a URI (either of which might contain a comma)
00393 could be safely carried in list-based field values like these:
00394
00395 Example-URIs: "http://example.com/a.html,foo",
00396               "http://without-a-comma.example.com/"
00397 Example-Dates: "Sat, 04 May 1996", "Wed, 14 Sep 2005"
00398 Note that double-quote delimiters are almost always used with the quoted-string
00399 production (Section 5.6.4); using a different syntax inside double-quotes will
00400 likely cause unnecessary confusion.
00401
00402 Many fields (such as Content-Type, defined in Section 8.3) use a common syntax
00403 for parameters that allows both unquoted (token) and quoted (quoted-string)
00404 syntax for a parameter value (Section 5.6.6). Use of common syntax allows
00405 recipients to reuse existing parser components. When allowing both forms, the
00406 meaning of a parameter value ought to be the same whether it was received as a
00407 token or a quoted string.
00408
00409 Note: For defining field value syntax, this specification uses an ABNF rule
00410 named after the field name to define the allowed grammar for that field's value
```

```
00411 (after said value has been extracted from the underlying messaging syntax and
00412 multiple instances combined into a list).
00413 */
```

7.36 src/http/HTTP.cpp File Reference

```
#include "HTTP.hpp"
```

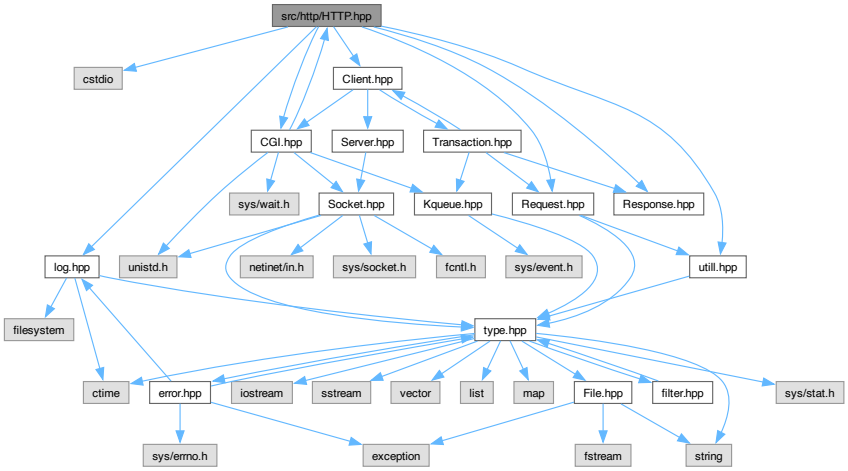
Include dependency graph for HTTP.cpp:



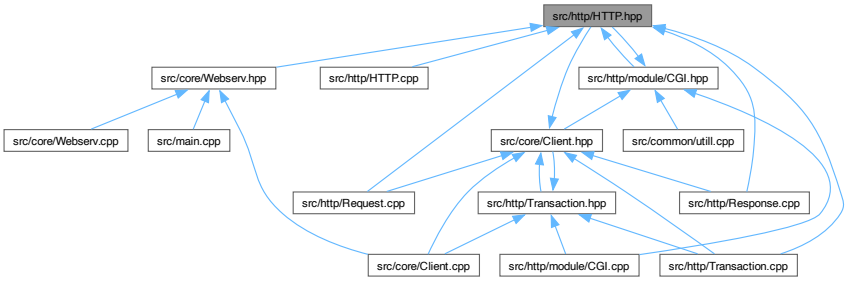
7.37 src/http/HTTP.hpp File Reference

```
#include <cstdio>
#include "log.hpp"
#include "utill.hpp"
#include "Client.hpp"
#include "CGI.hpp"
#include "Request.hpp"
#include "Response.hpp"
```

Include dependency graph for HTTP.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class HTTP

Macros

- #define CNT_METHOD 3
- #define CNT_VERSION 4
- #define CNT_ENCODING 2
- #define CNT_CONNECTION 2

Variables

- const str_t software = "webserv/1.0"
- const path_t dir_keys = "src/http/key/"
- const path_t file_status = dir_keys + "status.txt"
- const path_t file_mime = dir_keys + "mime.txt"
- const path_t file_header_in = dir_keys + "header_in.txt"
- const path_t file_header_out = dir_keys + "header_out.txt"
- const path_t file_environ = dir_keys + "environ.txt"

7.37.1 Macro Definition Documentation

7.37.1.1 CNT_CONNECTION

```
#define CNT_CONNECTION 2
```

7.37.1.2 CNT_ENCODING

```
#define CNT_ENCODING 2
```

7.37.1.3 CNT_METHOD

```
#define CNT_METHOD 3
```

7.37.1.4 CNT_VERSION

```
#define CNT_VERSION 4
```

7.37.2 Variable Documentation

7.37.2.1 dir_keys

```
const path_t dir_keys = "src/http/key/"
```

7.37.2.2 file_environ

```
const path_t file_environ = dir_keys + "environ.txt"
```

7.37.2.3 file_header_in

```
const path_t file_header_in = dir_keys + "header_in.txt"
```

7.37.2.4 file_header_out

```
const path_t file_header_out = dir_keys + "header_out.txt"
```

7.37.2.5 file_mime

```
const path_t file_mime = dir_keys + "mime.txt"
```

7.37.2.6 file_status

```
const path_t file_status = dir_keys + "status.txt"
```

7.37.2.7 software

```
const str_t software = "webserv/1.0"
```

7.38 HTTP.hpp

Go to the documentation of this file.

```
00001 #ifndef HTTP_HPP
00002 # define HTTP_HPP
00003
00004 # include <cstdio>
00005
00006 # include "log.hpp"
00007 # include "utill.hpp"
00008 # include "Client.hpp"
00009
00010 # define CNT_METHOD 3
00011 # define CNT_VERSION 4
00012 # define CNT_ENCODING 2
00013 # define CNT_CONNECTION 2
00014
00015 class Request;
00016 class Response;
00017
00018 const str_t        software        = "webserv/1.0";
00019
00020 const path_t  dir_keys        = "src/http/key/";
00021 const path_t  file_status      = dir_keys + "status.txt";
00022 const path_t  file_mime      = dir_keys + "mime.txt";
00023 const path_t  file_header_in    = dir_keys + "header_in.txt";
00024 const path_t  file_header_out  = dir_keys + "header_out.txt";
00025 const path_t  file_environ    = dir_keys + "environ.txt";
00026
00027 class HTTP {
00028     public:
00029         static http_t http;
00030         static keys_t key;
00031
00032         static void     init( void );
00033         static size_t   setConfig( const str_t&, const vec_config_t& );
00034         static size_t   setLocation( const path_t&, const vec_location_t& );
00035
00036         static void     GET( const path_t&, sstream_t&, size_t& );
00037         static void     POST( const Request& );
00038         static void     DELETE( const Request& );
00039
00040     private:
00041         static void     _assignHeader( void );
00042         static void     _assignStatus( void );
00043         static void     _assignMime( void );
00044         static void     _assignVec( vec_str_t &, const str_t[], size_t );
00045         static void     _initModule( void );
00046
00047         static bool     _setConfigMatchName( const str_t&, const vec_str_t&, const uint_t& );
00048 };
00049
00050 # include "CGI.hpp"
00051
00052 # include "Request.hpp"
00053 # include "Response.hpp"
00054
00055 #endif
00056
00057 /*
00058 9.3 GET
00059 The GET method means retrieve whatever information (in the form of an entity) is
00060 identified by the Request-URI. If the Request-URI refers to a data-producing
00061 process, it is the produced data which shall be returned as the entity in the
00062 response and not the source text of the process, unless that text happens to be
00063 the output of the process. The semantics of the GET method change to a
```

```
00064 “conditional GET” if the request message includes an If- Modified-Since,
00065 If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A
00066 conditional GET method requests that the entity be transferred only under the
00067 circumstances described by the conditional header field(s). The conditional GET
00068 method is intended to reduce unnecessary network usage by allowing cached
00069 entities to be refreshed without requiring multiple requests or transferring
00070 data already held by the client. The semantics of the GET method change to a
00071 “partial GET” if the request message includes a Range header field. A partial
00072 GET requests that only part of the entity be transferred, as described in
00073 section 14.35. The partial GET method is intended to reduce unnecessary network
00074 usage by allowing partially-retrieved entities to be completed without
00075 transferring data already held by the client. The response to a GET request is
00076 cacheable if and only if it meets the requirements for HTTP caching described in
00077 section 13. See section 15.1.3 for security considerations when used for forms.
00078
00079
00080 9.5 POST
00081 The POST method is used to request that the origin server accept the entity
00082 enclosed in the request as a new subordinate of the resource identified by the
00083 Request-URI in the Request-Line. POST is designed to allow a uniform method to
00084 cover the following functions: • Annotation of existing resources; • Posting a
00085 message to a bulletin board, newsgroup, mailing list, or similar group of
00086 articles; • Providing a block of data, such as the result of submitting a form,
00087 to a data-handling process; • Extending a database through an append operation.
00088 The actual function performed by the POST method is determined by the server and
00089 is usually dependent on the Request-URI.
00090
00091 The posted entity is subordinate to that URI in the same way that a file is
00092 subordinate to a directory containing it, a news article is subordinate to a
00093 newsgroup to which it is posted, or a record is subordinate to a database. The
00094 action performed by the POST method might not result in a resource that can be
00095 identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the
00096 appropriate response status, depending on whether or not the response includes
00097 an entity that describes the result. If a resource has been created on the
00098 origin server, the response SHOULD be 201 (Created) and contain an entity which
00099 describes the status of the request and refers to the new resource, and a
00100 Location header (see section 14.30). Responses to this method are not cacheable,
00101 unless the response includes appropriate Cache-Control or Expires header fields.
00102 However, the 303 (See Other) response can be used to direct the user agent to
00103 retrieve a cacheable resource. POST requests MUST obey the message transmission
00104 requirements set out in section 8.2. See section 15.1.3 for security
00105 considerations.
00106
00107
00108 9.7 DELETE
00109 The DELETE method requests that the origin server delete the resource identified
00110 by the Request-URI. This method MAY be overridden by human intervention (or
00111 other means) on the origin server. The client cannot be guaranteed that the
00112 operation has been carried out, even if the status code returned from the origin
00113 server indicates that the action has been completed successfully. However, the
00114 server SHOULD NOT indicate success unless, at the time the response is given, it
00115 intends to delete the resource or move it to an inaccessible location. A
00116 successful response SHOULD be 200 (OK) if the response includes an entity
00117 describing the status, 202 (Accepted) if the action has not yet been enacted, or
00118 204 (No Content) if the action has been enacted but the response does not
00119 include an entity. If the request passes through a cache and the Request-URI
00120 identifies one or more currently cached entities, those entries SHOULD be
00121 treated as stale. Responses to this method are not cacheable.
00122
00123 The scheme and host are case-insensitive and normally provided in lowercase; all
00124 other components are compared in a case-sensitive manner.
00125
00126 For example, the following three URIs are equivalent:
00127
00128    http://example.com:80/~smith/home.html
00129    http://EXAMPLE.com/%7Esmith/home.html
00130    http://EXAMPLE.com:/%7esmith/home.html
00131
00132
00133 4.3.1. URI Origin
00134 The origin for a given URI is the triple of scheme, host, and port after
00135 normalizing the scheme and host to lowercase and normalizing the port to remove
00136 any leading zeros. If port is elided from the URI, the default port for that
00137 scheme is used. For example, the URI
00138
00139    https://Example.Com/happy.js
00140 would have the origin
00141
00142    { "https", "example.com", "443" }
00143
00144
00145 5.2. Field Lines and Combined Field Value
00146 Field sections are composed of any number of field lines, each with a field name
00147 (see Section 5.1) identifying the field, and a field line value that conveys
00148 data for that instance of the field.
00149
00150 When a field name is only present once in a section, the combined field value
```

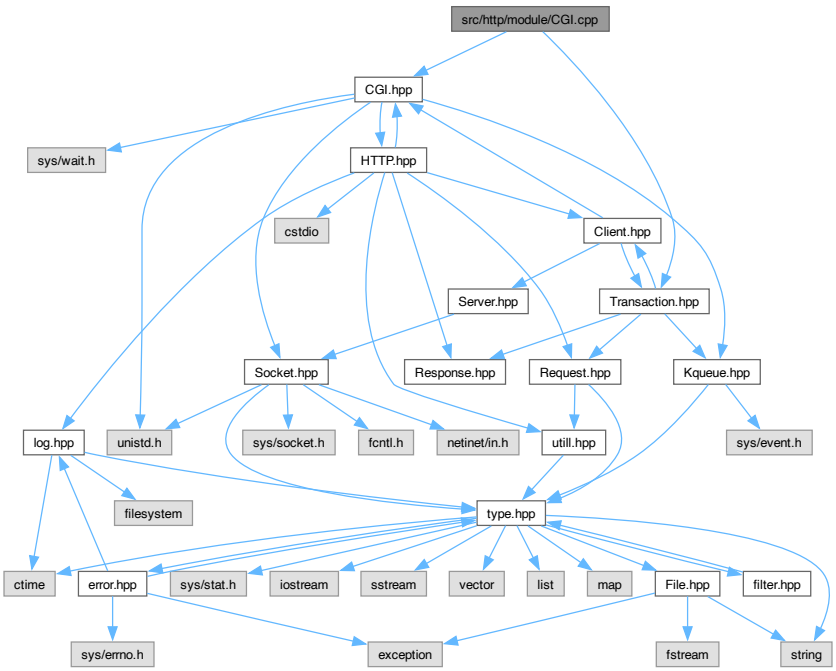
```
00151 for that field consists of the corresponding field line value. When a field name
00152 is repeated within a section, its combined field value consists of the list of
00153 corresponding field line values within that section, concatenated in order, with
00154 each field line value separated by a comma.
00155
00156 For example, this section:
00157
00158 Example-Field: Foo, Bar
00159 Example-Field: Baz
00160 contains two field lines, both with the field name "Example-Field". The first
00161 field line has a field line value of "Foo, Bar", while the second field line
00162 value is "Baz". The field value for "Example-Field" is the list "Foo, Bar, Baz".
00163
00164 A server MUST NOT apply a request to the target resource until it receives the
00165 entire request header section, since later header field lines might include
00166 conditionals, authentication credentials, or deliberately misleading duplicate
00167 header fields that could impact request processing.
00168
00169
00170 5.4. Field Limits
00171 HTTP does not place a predefined limit on the length of each field line, field
00172 value, or on the length of a header or trailer section as a whole, as described
00173 in Section 2. Various ad hoc limitations on individual lengths are found in
00174 practice, often depending on the specific field's semantics.
00175
00176 A server that receives a request header field line, field value, or set of
00177 fields larger than it wishes to process MUST respond with an appropriate 4xx
00178 (Client Error) status code. Ignoring such header fields would increase the
00179 server's vulnerability to request smuggling attacks (Section 11.2 of
00180 [HTTP/1.1]).
00181
00182 A client MAY discard or truncate received field lines that are larger than the
00183 client wishes to process if the field semantics are such that the dropped
00184 value(s) can be safely ignored without changing the message framing or response
00185 semantics.
00186 */
00187
00188 /*
00189 Syntax: location [ = | ~ | ~* | ^~ ] uri { ... }
00190 location @name { ... }
00191 Default:    --
00192 Context:    server, location
00193 Sets configuration depending on a request URI.
00194
00195 The matching is performed against a normalized URI, after decoding the text
00196 encoded in the "%XX" form, resolving references to relative path components
00197 "." and "..", and possible compression of two or more adjacent slashes into a
00198 single slash.
00199
00200 A location can either be defined by a prefix string, or by a regular expression.
00201 Regular expressions are specified with the preceding "~*" modifier
00202 (for case-insensitive matching), or the "~" modifier (for case-sensitive matching).
00203 To find location matching a given request, nginx first checks locations defined
00204 using the prefix strings (prefix locations). Among them, the location with the
00205 longest matching prefix is selected and remembered. Then regular expressions are
00206 checked, in the order of their appearance in the configuration file. The search
00207 of regular expressions terminates on the first match, and the corresponding
00208 configuration is used. If no match with a regular expression is found then the
00209 configuration of the prefix location remembered earlier is used.
00210
00211 location blocks can be nested, with some exceptions mentioned below.
00212
00213 For case-insensitive operating systems such as macOS and Cygwin, matching with
00214 prefix strings ignores a case (0.7.7). However, comparison is limited
00215 to one-byte locales.
00216
00217 Regular expressions can contain captures (0.7.40) that can later be used
00218 in other directives.
00219
00220 If the longest matching prefix location has the "^~" modifier then regular
00221 expressions are not checked.
00222
00223 Also, using the "=" modifier it is possible to define an exact match of
00224 URI and location. If an exact match is found, the search terminates.
00225 For example, if a "/" request happens frequently, defining
00226 "location = /" will speed up the processing of these requests, as search
00227 terminates right after the first comparison. Such a location cannot obviously
00228 contain nested locations.
00229
00230 In versions from 0.7.1 to 0.8.41, if a request matched the prefix location
00231 without the "=" and "^~" modifiers, the search also terminated and regular
00232 expressions were not checked.
00233 Let's illustrate the above by an example:
00234
00235 location = / {
00236     [ configuration A ]
00237 }
```

```
00238
00239 location / {
00240     [ configuration B ]
00241 }
00242
00243 location /documents/ {
00244     [ configuration C ]
00245 }
00246
00247 location ^~ /images/ {
00248     [ configuration D ]
00249 }
00250
00251 location ~* \.(gif|jpg|jpeg)$ {
00252     [ configuration E ]
00253 }
00254 The “/” request will match configuration A, the “/index.html” request will
00255 match configuration B, the “/documents/document.html” request will match
00256 configuration C, the “/images/1.gif” request will match configuration D,
00257 and the “/documents/1.jpg” request will match configuration E.
00258
00259 The “@” prefix defines a named location. Such a location is not used for a
00260 regular request processing, but instead used for request redirection.
00261 They cannot be nested, and cannot contain nested locations.
00262
00263 If a location is defined by a prefix string that ends with the slash
00264 character, and requests are processed by one of proxy_pass, fastcgi_pass,
00265 uwsgi_pass, scgi_pass, memcached_pass, or grpc_pass, then the special
00266 processing is performed. In response to a request with URI equal to this
00267 string, but without the trailing slash, a permanent redirect with the code
00268 301 will be returned to the requested URI with the slash appended. If this
00269 is not desired, an exact match of the URI and location could be defined
00270 like this:
00271
00272 location /user/ {
00273     proxy_pass http://user.example.com;
00274 }
00275
00276 location = /user {
00277     proxy_pass http://login.example.com;
00278 }
00279 */
00280
```

7.39 src/http/module/CGI.cpp File Reference

```
#include "CGI.hpp"
#include "Transaction.hpp"
```

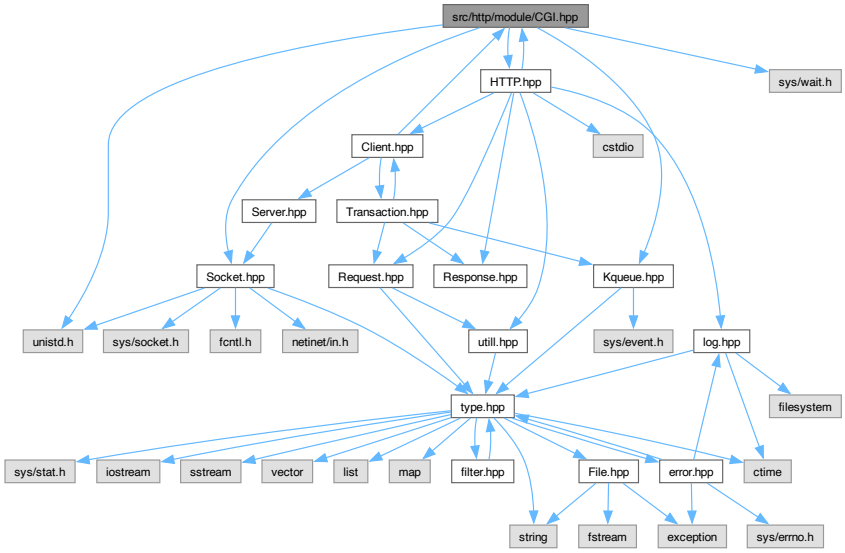
Include dependency graph for CGI.cpp:



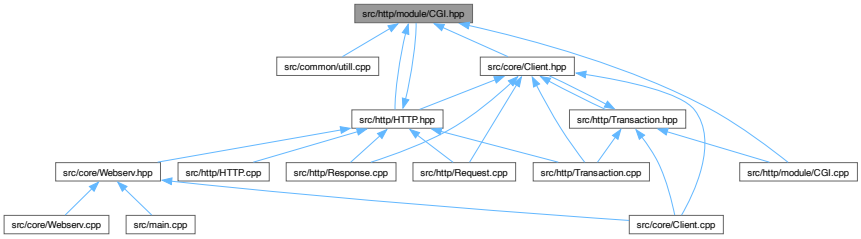
7.40 src/http/module/CGI.hpp File Reference

```
#include <unistd.h>
#include <sys/wait.h>
#include "HTTP.hpp"
#include "Socket.hpp"
#include "Kqueue.hpp"
```

Include dependency graph for CGI.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct c_buffer_s
- class CGI

Macros

- #define SIZE_BUFF_C 2048

Typedefs

- typedef struct c_buffer_s c_buffer_t
- typedef struct message_s message_t

Enumerations

- enum pipe_mode_e { R , W }

7.40.1 Macro Definition Documentation

7.40.1.1 SIZE_BUFF_C

```
#define SIZE_BUFF_C 2048
```

7.40.2 Typedef Documentation

7.40.2.1 c_buffer_t

```
typedef struct c_buffer_s c_buffer_t
```

7.40.2.2 message_t

```
typedef struct message_s message_t
```

7.40.3 Enumeration Type Documentation

7.40.3.1 pipe_mode_e

```
enum pipe_mode_e
```

Enumerator

R	
W	

7.41 CGI.hpp

Go to the documentation of this file.

```
00001 #ifndef CGI_HPP
00002 # define CGI_HPP
00003
00004 # include <unistd.h>
00005 # include <sys/wait.h>
00006
00007 # include "HTTP.hpp"
00008 # include "Socket.hpp"
00009 # include "Kqueue.hpp"
00010
00011 # define SIZE_BUFF_C 2048
00012
00013 enum pipe_mode_e { R, W };
00014
00015 class Request;
00016
00017 typedef struct c_buffer_s {
```

```
00018     c_buffer_s( void );
00019
00020     char        ptr[SIZE_BUFF_C];
00021
00022     size_t      total;
00023     ssize_t     read;
00024 }   c_buffer_t;
00025
00026 typedef struct message_s message_t;
00027
00028 class CGI {
00029     public:
00030         static map_str_path_t script_bin;
00031         static map_uint_str_t environ_list;
00032
00033         static void             init( void );
00034         static void             detach( const Request&, process_t& );
00035
00036         static void             proceedParent( pid_t, const fd_t&, Kqueue& );
00037         static void             write( const process_t&, const char*, const size_t& );
00038         static void             read( const process_t&, sstream_t& );
00039         static void             wait( process_t& );
00040         static void             build( message_t& );
00041
00042         static void             proceedChild( const Request&, process_t& );
00043
00044     private:
00045         /* init */
00046         static void             _assignScriptBin( void );
00047         static void             _assignEnvironList( void );
00048         static void             _assignVectorChar( vec_cstr_t&, const vec_str_t& );
00049
00050         /* detach */
00051         static void             _valid( const Request& );
00052
00053         /* Parent */
00054         static void             _buildLine( message_t& );
00055         static void             _buildHeader( message_t& );
00056         static void             _buildHeaderServer( message_t& );
00057         static void             _buildHeaderType( message_t&, const size_t& );
00058         static void             _buildHeaderLen( message_t&, const size_t& );
00059         static void             _buildChunk( message_t& );
00060
00061         /* Child */
00062         static void             _buildEnviron( const Request&, process_t& );
00063         static bool             _buildEnvironVar( const Request&, process_t&, uint_t idx );
00064
00065         static void             _execve( const process_t& );
00066         static bool             _redirect( const process_t& );
00067 };
00068
00069 #endif
00070
00071 /*
00072 3.  Invoking the Script
00073
00074 3.1.  Server Responsibilities
00075
00076    The server acts as an application gateway.  It receives the request
00077    from the client, selects a CGI script to handle the request, converts
00078    the client request to a CGI request, executes the script and converts
00079    the CGI response into a response for the client.  When processing the
00080    client request, it is responsible for implementing any protocol or
00081    transport level authentication and security.  The server MAY also
00082    function in a 'non-transparent' manner, modifying the request or
00083    response in order to provide some additional service, such as media
00084    type transformation or protocol reduction.
00085
00086    The server MUST perform translations and protocol conversions on the
00087    client request data required by this specification.  Furthermore, the
00088    server retains its responsibility to the client to conform to the
00089    relevant network protocol even if the CGI script fails to conform to
00090    this specification.
00091
00092    If the server is applying authentication to the request, then it MUST
00093    NOT execute the script unless the request passes all defined access
00094    controls.
00095
00096
00097
00098
00099
00100
00101
00102
00103 Robinson & Coar              Informational                      [Page 8]
00104
```

```
00105 RFC 3875                  CGI Version 1.1                October 2004
00106
00107
00108 3.2.  Script Selection
00109
00110    The server determines which CGI is script to be executed based on a
00111    generic-form URI supplied by the client.  This URI includes a
00112    hierarchical path with components separated by "/".  For any
00113    particular request, the server will identify all or a leading part of
00114    this path with an individual script, thus placing the script at a
00115    particular point in the path hierarchy.  The remainder of the path,
00116    if any, is a resource or sub-resource identifier to be interpreted by
00117    the script.
00118
00119    Information about this split of the path is available to the script
00120    in the meta-variables, described below.  Support for non-hierarchical
00121    URI schemes is outside the scope of this specification.
00122
00123 3.3.  The Script-URI
00124
00125    The mapping from client request URI to choice of script is defined by
00126    the particular server implementation and its configuration.  The
00127    server may allow the script to be identified with a set of several
00128    different URI path hierarchies, and therefore is permitted to replace
00129    the URI by other members of this set during processing and generation
00130    of the meta-variables.  The server
00131
00132       1. MAY preserve the URI in the particular client request; or
00133
00134       2. it MAY select a canonical URI from the set of possible values
00135          for each script; or
00136
00137       3. it can implement any other selection of URI from the set.
00138
00139    From the meta-variables thus generated, a URI, the 'Script-URI', can
00140    be constructed.  This MUST have the property that if the client had
00141    accessed this URI instead, then the script would have been executed
00142    with the same values for the SCRIPT_NAME, PATH_INFO and QUERY_STRING
00143    meta-variables.  The Script-URI has the structure of a generic URI as
00144    defined in section 3 of RFC 2396 [2], with the exception that object
00145    parameters and fragment identifiers are not permitted.  The various
00146    components of the Script-URI are defined by some of the
00147    meta-variables (see below);
00148
00149       script-URI = <scheme> "://" <server-name> ":" <server-port>
00150                    <script-path> <extra-path> "?" <query-string>
00151
00152    where <scheme> is found from SERVER_PROTOCOL, <server-name>,
00153    <server-port> and <query-string> are the values of the respective
00154    meta-variables.  The SCRIPT_NAME and PATH_INFO values, URL-encoded
00155    with ";", "=" and "?"  reserved, give <script-path> and <extra-path>.
00156
00157
00158
00159 Robinson & Coar              Informational                     [Page 9]
00160
00161 RFC 3875                  CGI Version 1.1                October 2004
00162
00163
00164    See section 4.1.5 for more information about the PATH_INFO
00165    meta-variable.
00166
00167    The scheme and the protocol are not identical as the scheme
00168    identifies the access method in addition to the application protocol.
00169    For example, a resource accessed using Transport Layer Security (TLS)
00170    [14] would have a request URI with a scheme of https when using the
00171    HTTP protocol [19].  CGI/1.1 provides no generic means for the script
00172    to reconstruct this, and therefore the Script-URI as defined includes
00173    the base protocol used.  However, a script MAY make use of
00174    scheme-specific meta-variables to better deduce the URI scheme.
00175
00176    Note that this definition also allows URIs to be constructed which
00177    would invoke the script with any permitted values for the path-info
00178    or query-string, by modifying the appropriate components.
00179
00180 3.4.  Execution
00181
00182    The script is invoked in a system-defined manner.  Unless specified
00183    otherwise, the file containing the script will be invoked as an
00184    executable program.  The server prepares the CGI request as described
00185    in section 4; this comprises the request meta-variables (immediately
00186    available to the script on execution) and request message data.  The
00187    request data need not be immediately available to the script; the
00188    script can be executed before all this data has been received by the
00189    server from the client.  The response from the script is returned to
00190    the server as described in sections 5 and 6.
00191
```

```
00192    In the event of an error condition, the server can interrupt or
00193    terminate script execution at any time and without warning.  That
00194    could occur, for example, in the event of a transport failure between
00195    the server and the client; so the script SHOULD be prepared to handle
00196    abnormal termination.
00197
00198 4.  The CGI Request
00199
00200    Information about a request comes from two different sources; the
00201    request meta-variables and any associated message-body.
00202
00203 4.1.  Request Meta-Variables
00204
00205    Meta-variables contain data about the request passed from the server
00206    to the script, and are accessed by the script in a system-defined
00207    manner.  Meta-variables are identified by case-insensitive names;
00208    there cannot be two different variables whose names differ in case
00209    only.  Here they are shown using a canonical representation of
00210    capitals plus underscore ("_").  A particular system can define a
00211    different representation.
00212
00213
00214
00215 Robinson & Coar              Informational                    [Page 10]
00216
00217 RFC 3875                    CGI Version 1.1                October 2004
00218
00219
00220       meta-variable-name = "AUTH_TYPE" | "CONTENT_LENGTH" |
00221                            "CONTENT_TYPE" | "GATEWAY_INTERFACE" |
00222                            "PATH_INFO" | "PATH_TRANSLATED" |
00223                            "QUERY_STRING" | "REMOTE_ADDR" |
00224                            "REMOTE_HOST" | "REMOTE_IDENT" |
00225                            "REMOTE_USER" | "REQUEST_METHOD" |
00226                            "SCRIPT_NAME" | "SERVER_NAME" |
00227                            "SERVER_PORT" | "SERVER_PROTOCOL" |
00228                            "SERVER_SOFTWARE" | scheme |
00229                           protocol-var-name | extension-var-name
00230       protocol-var-name  = ( protocol | scheme ) "_" var-name
00231       scheme             = alpha *( alpha | digit | "+" | "-" | "." )
00232       var-name           = token
00233       extension-var-name = token
00234
00235    Meta-variables with the same name as a scheme, and names beginning
00236    with the name of a protocol or scheme (e.g., HTTP_ACCEPT) are also
00237    defined.  The number and meaning of these variables may change
00238    independently of this specification.  (See also section 4.1.18.)
00239
00240    The server MAY set additional implementation-defined extension meta-
00241    variables, whose names SHOULD be prefixed with "X_".
00242
00243    This specification does not distinguish between zero-length (NULL)
00244    values and missing values.  For example, a script cannot distinguish
00245    between the two requests http://host/script and http://host/script?
00246    as in both cases the QUERY_STRING meta-variable would be NULL.
00247
00248       meta-variable-value = "" | 1*<TEXT, CHAR or tokens of value>
00249
00250    An optional meta-variable may be omitted (left unset) if its value is
00251    NULL.  Meta-variable values MUST be considered case-sensitive except
00252    as noted otherwise.  The representation of the characters in the
00253    meta-variables is system-defined; the server MUST convert values to
00254    that representation.
00255
00256 4.1.1.  AUTH_TYPE
00257
00258    The AUTH_TYPE variable identifies any mechanism used by the server to
00259    authenticate the user.  It contains a case-insensitive value defined
00260    by the client protocol or server implementation.
00261
00262    For HTTP, if the client request required authentication for external
00263    access, then the server MUST set the value of this variable from the
00264    'auth-scheme' token in the request Authorization header field.
00265
00266
00267
00268
00269
00270
00271 Robinson & Coar              Informational                    [Page 11]
00272
00273 RFC 3875                    CGI Version 1.1                October 2004
00274
00275
00276       AUTH_TYPE      = "" | auth-scheme
00277       auth-scheme    = "Basic" | "Digest" | extension-auth
00278       extension-auth = token
```

```
00279
00280    HTTP access authentication schemes are described in RFC 2617 [5].
00281
00282 4.1.2.  CONTENT_LENGTH
00283
00284    The CONTENT_LENGTH variable contains the size of the message-body
00285    attached to the request, if any, in decimal number of octets.  If no
00286    data is attached, then NULL (or unset).
00287
00288       CONTENT_LENGTH = "" | 1*digit
00289
00290    The server MUST set this meta-variable if and only if the request is
00291    accompanied by a message-body entity.  The CONTENT_LENGTH value must
00292    reflect the length of the message-body after the server has removed
00293    any transfer-codings or content-codings.
00294
00295 4.1.3.  CONTENT_TYPE
00296
00297    If the request includes a message-body, the CONTENT_TYPE variable is
00298    set to the Internet Media Type [6] of the message-body.
00299
00300       CONTENT_TYPE = "" | media-type
00301       media-type   = type "/" subtype *( ";" parameter )
00302       type         = token
00303       subtype      = token
00304       parameter    = attribute "=" value
00305       attribute    = token
00306       value        = token | quoted-string
00307
00308    The type, subtype and parameter attribute names are not
00309    case-sensitive.  Parameter values may be case sensitive.  Media types
00310    and their use in HTTP are described section 3.7 of the HTTP/1.1
00311    specification [4].
00312
00313    There is no default value for this variable.  If and only if it is
00314    unset, then the script MAY attempt to determine the media type from
00315    the data received.  If the type remains unknown, then the script MAY
00316    choose to assume a type of application/octet-stream or it may reject
00317    the request with an error (as described in section 6.3.3).
00318
00319    Each media-type defines a set of optional and mandatory parameters.
00320    This may include a charset parameter with a case-insensitive value
00321    defining the coded character set for the message-body.  If the
00322
00323
00324
00325
00326
00327 Robinson & Coar              Informational                    [Page 12]
00328
00329 RFC 3875                    CGI Version 1.1                October 2004
00330
00331
00332    charset parameter is omitted, then the default value should be
00333    derived according to whichever of the following rules is the first to
00334    apply:
00335
00336       1. There MAY be a system-defined default charset for some
00337          media-types.
00338
00339       2. The default for media-types of type "text" is ISO-8859-1 [4].
00340
00341       3. Any default defined in the media-type specification.
00342
00343       4. The default is US-ASCII.
00344
00345    The server MUST set this meta-variable if an HTTP Content-Type field
00346    is present in the client request header.  If the server receives a
00347    request with an attached entity but no Content-Type header field, it
00348    MAY attempt to determine the correct content type, otherwise it
00349    should omit this meta-variable.
00350
00351 4.1.4.  GATEWAY_INTERFACE
00352
00353    The GATEWAY_INTERFACE variable MUST be set to the dialect of CGI
00354    being used by the server to communicate with the script.  Syntax:
00355
00356       GATEWAY_INTERFACE = "CGI" "/" 1*digit "." 1*digit
00357
00358    Note that the major and minor numbers are treated as separate
00359    integers and hence each may be incremented higher than a single
00360    digit.  Thus CGI/2.4 is a lower version than CGI/2.13 which in turn
00361    is lower than CGI/12.3.  Leading zeros MUST be ignored by the script
00362    and MUST NOT be generated by the server.
00363
00364    This document defines the 1.1 version of the CGI interface.
00365
```

```
00366 4.1.5.  PATH_INFO
00367
00368    The PATH_INFO variable specifies a path to be interpreted by the CGI
00369    script.  It identifies the resource or sub-resource to be returned by
00370    the CGI script, and is derived from the portion of the URI path
00371    hierarchy following the part that identifies the script itself.
00372    Unlike a URI path, the PATH_INFO is not URL-encoded, and cannot
00373    contain path-segment parameters.  A PATH_INFO of "/" represents a
00374    single void path segment.
00375
00376       PATH_INFO = "" | ( "/" path )
00377       path      = lsegment *( "/" lsegment )
00378       lsegment  = *lchar
00379       lchar     = <any TEXT or CTL except "/">
00380
00381
00382
00383 Robinson & Coar              Informational                     [Page 13]
00384
00385 RFC 3875                    CGI Version 1.1                 October 2004
00386
00387
00388    The value is considered case-sensitive and the server MUST preserve
00389    the case of the path as presented in the request URI.  The server MAY
00390    impose restrictions and limitations on what values it permits for
00391    PATH_INFO, and MAY reject the request with an error if it encounters
00392    any values considered objectionable.  That MAY include any requests
00393    that would result in an encoded "/" being decoded into PATH_INFO, as
00394    this might represent a loss of information to the script.  Similarly,
00395    treatment of non US-ASCII characters in the path is system-defined.
00396
00397    URL-encoded, the PATH_INFO string forms the extra-path component of
00398    the Script-URI (see section 3.3) which follows the SCRIPT_NAME part
00399    of that path.
00400
00401 4.1.6.  PATH_TRANSLATED
00402
00403    The PATH_TRANSLATED variable is derived by taking the PATH_INFO
00404    value, parsing it as a local URI in its own right, and performing any
00405    virtual-to-physical translation appropriate to map it onto the
00406    server's document repository structure.  The set of characters
00407    permitted in the result is system-defined.
00408
00409       PATH_TRANSLATED = *<any character>
00410
00411    This is the file location that would be accessed by a request for
00412
00413       <scheme> "://" <server-name> ":" <server-port> <extra-path>
00414
00415    where <scheme> is the scheme for the original client request and
00416    <extra-path> is a URL-encoded version of PATH_INFO, with ";", "=" and
00417    "?"  reserved.  For example, a request such as the following:
00418
00419       http://somehost.com/cgi-bin/somescript/this%2eis%2epath%3binfo
00420
00421    would result in a PATH_INFO value of
00422
00423       /this.is.the.path;info
00424
00425    An internal URI is constructed from the scheme, server location and
00426    the URL-encoded PATH_INFO:
00427
00428       http://somehost.com/this.is.the.path%3binfo
00429
00430    This would then be translated to a location in the server's document
00431    repository, perhaps a filesystem path something like this:
00432
00433       /usr/local/www/htdocs/this.is.the.path;info
00434
00435    The value of PATH_TRANSLATED is the result of the translation.
00436
00437
00438
00439 Robinson & Coar              Informational                     [Page 14]
00440
00441 RFC 3875                    CGI Version 1.1                 October 2004
00442
00443
00444    The value is derived in this way irrespective of whether it maps to a
00445    valid repository location.  The server MUST preserve the case of the
00446    extra-path segment unless the underlying repository supports case-
00447    insensitive names.  If the repository is only case-aware, case-
00448    preserving, or case-blind with regard to document names, the server
00449    is not required to preserve the case of the original segment through
00450    the translation.
00451
00452    The translation algorithm the server uses to derive PATH_TRANSLATED
```

```
00453    is implementation-defined; CGI scripts which use this variable may
00454    suffer limited portability.
00455
00456    The server SHOULD set this meta-variable if the request URI includes
00457    a path-info component.  If PATH_INFO is NULL, then the
00458    PATH_TRANSLATED variable MUST be set to NULL (or unset).
00459
00460 4.1.7.  QUERY_STRING
00461
00462    The QUERY_STRING variable contains a URL-encoded search or parameter
00463    string; it provides information to the CGI script to affect or refine
00464    the document to be returned by the script.
00465
00466    The URL syntax for a search string is described in section 3 of RFC
00467    2396 [2].  The QUERY_STRING value is case-sensitive.
00468
00469       QUERY_STRING = query-string
00470       query-string = *uric
00471       uric         = reserved | unreserved | escaped
00472
00473    When parsing and decoding the query string, the details of the
00474    parsing, reserved characters and support for non US-ASCII characters
00475    depends on the context.  For example, form submission from an HTML
00476    document [18] uses application/x-www-form-urlencoded encoding, in
00477    which the characters "+", "&" and "=" are reserved, and the ISO
00478    8859-1 encoding may be used for non US-ASCII characters.
00479
00480    The QUERY_STRING value provides the query-string part of the
00481    Script-URI.  (See section 3.3).
00482
00483    The server MUST set this variable; if the Script-URI does not include
00484    a query component, the QUERY_STRING MUST be defined as an empty
00485    string ("").
00486
00487 4.1.8.  REMOTE_ADDR
00488
00489    The REMOTE_ADDR variable MUST be set to the network address of the
00490    client sending the request to the server.
00491
00492
00493
00494
00495 Robinson & Coar              Informational                     [Page 15]
00496
00497 RFC 3875                   CGI Version 1.1                October 2004
00498
00499
00500       REMOTE_ADDR  = hostnumber
00501       hostnumber   = ipv4-address | ipv6-address
00502       ipv4-address = 1*3digit "." 1*3digit "." 1*3digit "." 1*3digit
00503       ipv6-address = hexpart [ ":" ipv4-address ]
00504       hexpart      = hexseq | ( [ hexseq ] "::" [ hexseq ] )
00505       hexseq       = 1*4hex *( ":" 1*4hex )
00506
00507    The format of an IPv6 address is described in RFC 3513 [15].
00508
00509 4.1.9.  REMOTE_HOST
00510
00511    The REMOTE_HOST variable contains the fully qualified domain name of
00512    the client sending the request to the server, if available, otherwise
00513    NULL.  Fully qualified domain names take the form as described in
00514    section 3.5 of RFC 1034 [17] and section 2.1 of RFC 1123 [12].
00515    Domain names are not case sensitive.
00516
00517       REMOTE_HOST   = "" | hostname | hostnumber
00518       hostname      = *( domainlabel "." ) toplabel [ "." ]
00519       domainlabel   = alphanum [ *alphahypdigit alphanum ]
00520       toplabel      = alpha [ *alphahypdigit alphanum ]
00521       alphahypdigit = alphanum | "-"
00522
00523    The server SHOULD set this variable.  If the hostname is not
00524    available for performance reasons or otherwise, the server MAY
00525    substitute the REMOTE_ADDR value.
00526
00527 4.1.10.  REMOTE_IDENT
00528
00529    The REMOTE_IDENT variable MAY be used to provide identity information
00530    reported about the connection by an RFC 1413 [20] request to the
00531    remote agent, if available.  The server may choose not to support
00532    this feature, or not to request the data for efficiency reasons, or
00533    not to return available identity data.
00534
00535       REMOTE_IDENT = *TEXT
00536
00537    The data returned may be used for authentication purposes, but the
00538    level of trust reposed in it should be minimal.
00539
```

```
00540 4.1.11.  REMOTE_USER
00541
00542    The REMOTE_USER variable provides a user identification string
00543    supplied by client as part of user authentication.
00544
00545       REMOTE_USER = *TEXT
00546
00547
00548
00549
00550
00551 Robinson & Coar              Informational                     [Page 16]
00552
00553 RFC 3875                    CGI Version 1.1                 October 2004
00554
00555
00556    If the client request required HTTP Authentication [5] (e.g., the
00557    AUTH_TYPE meta-variable is set to "Basic" or "Digest"), then the
00558    value of the REMOTE_USER meta-variable MUST be set to the user-ID
00559    supplied.
00560
00561 4.1.12.  REQUEST_METHOD
00562
00563    The REQUEST_METHOD meta-variable MUST be set to the method which
00564    should be used by the script to process the request, as described in
00565    section 4.3.
00566
00567       REQUEST_METHOD   = method
00568       method           = "GET" | "POST" | "HEAD" | extension-method
00569       extension-method = "PUT" | "DELETE" | token
00570
00571    The method is case sensitive.  The HTTP methods are described in
00572    section 5.1.1 of the HTTP/1.0 specification [1] and section 5.1.1 of
00573    the HTTP/1.1 specification [4].
00574
00575 4.1.13.  SCRIPT_NAME
00576
00577    The SCRIPT_NAME variable MUST be set to a URI path (not URL-encoded)
00578    which could identify the CGI script (rather than the script's
00579    output).  The syntax is the same as for PATH_INFO (section 4.1.5)
00580
00581       SCRIPT_NAME = "" | ( "/" path )
00582
00583    The leading "/" is not part of the path.  It is optional if the path
00584    is NULL; however, the variable MUST still be set in that case.
00585
00586    The SCRIPT_NAME string forms some leading part of the path component
00587    of the Script-URI derived in some implementation-defined manner.  No
00588    PATH_INFO segment (see section 4.1.5) is included in the SCRIPT_NAME
00589    value.
00590
00591 4.1.14.  SERVER_NAME
00592
00593    The SERVER_NAME variable MUST be set to the name of the server host
00594    to which the client request is directed.  It is a case-insensitive
00595    hostname or network address.  It forms the host part of the
00596    Script-URI.
00597
00598       SERVER_NAME = server-name
00599       server-name = hostname | ipv4-address | ( "[" ipv6-address "]" )
00600
00601
00602
00603
00604
00605
00606
00607 Robinson & Coar              Informational                     [Page 17]
00608
00609 RFC 3875                    CGI Version 1.1                 October 2004
00610
00611
00612    A deployed server can have more than one possible value for this
00613    variable, where several HTTP virtual hosts share the same IP address.
00614    In that case, the server would use the contents of the request's Host
00615    header field to select the correct virtual host.
00616
00617 4.1.15.  SERVER_PORT
00618
00619    The SERVER_PORT variable MUST be set to the TCP/IP port number on
00620    which this request is received from the client.  This value is used
00621    in the port part of the Script-URI.
00622
00623       SERVER_PORT = server-port
00624       server-port = 1*digit
00625
00626    Note that this variable MUST be set, even if the port is the default
```

```
00627    port for the scheme and could otherwise be omitted from a URI.
00628
00629 4.1.16.  SERVER_PROTOCOL
00630
00631    The SERVER_PROTOCOL variable MUST be set to the name and version of
00632    the application protocol used for this CGI request.  This MAY differ
00633    from the protocol version used by the server in its communication
00634    with the client.
00635
00636       SERVER_PROTOCOL   = HTTP-Version | "INCLUDED" | extension-version
00637       HTTP-Version      = "HTTP" "/" 1*digit "." 1*digit
00638       extension-version = protocol [ "/" 1*digit "." 1*digit ]
00639       protocol          = token
00640
00641    Here, 'protocol' defines the syntax of some of the information
00642    passing between the server and the script (the 'protocol-specific'
00643    features).  It is not case sensitive and is usually presented in
00644    upper case.  The protocol is not the same as the scheme part of the
00645    script URI, which defines the overall access mechanism used by the
00646    client to communicate with the server.  For example, a request that
00647    reaches the script with a protocol of "HTTP" may have used an "https"
00648    scheme.
00649
00650    A well-known value for SERVER_PROTOCOL which the server MAY use is
00651    "INCLUDED", which signals that the current document is being included
00652    as part of a composite document, rather than being the direct target
00653    of the client request.  The script should treat this as an HTTP/1.0
00654    request.
00655
00656
00657
00658
00659
00660
00661
00662
00663 Robinson & Coar              Informational                    [Page 18]
00664
00665 RFC 3875                   CGI Version 1.1                October 2004
00666
00667
00668 4.1.17.  SERVER_SOFTWARE
00669
00670    The SERVER_SOFTWARE meta-variable MUST be set to the name and version
00671    of the information server software making the CGI request (and
00672    running the gateway).  It SHOULD be the same as the server
00673    description reported to the client, if any.
00674
00675       SERVER_SOFTWARE = 1*( product | comment )
00676       product         = token [ "/" product-version ]
00677       product-version = token
00678       comment         = "(" *( ctext | comment ) ")"
00679       ctext           = <any TEXT excluding "(" and ")">
00680
00681 4.1.18.  Protocol-Specific Meta-Variables
00682
00683    The server SHOULD set meta-variables specific to the protocol and
00684    scheme for the request.  Interpretation of protocol-specific
00685    variables depends on the protocol version in SERVER_PROTOCOL.  The
00686    server MAY set a meta-variable with the name of the scheme to a
00687    non-NULL value if the scheme is not the same as the protocol.  The
00688    presence of such a variable indicates to a script which scheme is
00689    used by the request.
00690
00691    Meta-variables with names beginning with "HTTP_" contain values read
00692    from the client request header fields, if the protocol used is HTTP.
00693    The HTTP header field name is converted to upper case, has all
00694    occurrences of "-" replaced with "_" and has "HTTP_" prepended to
00695    give the meta-variable name.  The header data can be presented as
00696    sent by the client, or can be rewritten in ways which do not change
00697    its semantics.  If multiple header fields with the same field-name
00698    are received then the server MUST rewrite them as a single value
00699    having the same semantics.  Similarly, a header field that spans
00700    multiple lines MUST be merged onto a single line.  The server MUST,
00701    if necessary, change the representation of the data (for example, the
00702    character set) to be appropriate for a CGI meta-variable.
00703
00704    The server is not required to create meta-variables for all the
00705    header fields that it receives.  In particular, it SHOULD remove any
00706    header fields carrying authentication information, such as
00707    'Authorization'; or that are available to the script in other
00708    variables, such as 'Content-Length' and 'Content-Type'.  The server
00709    MAY remove header fields that relate solely to client-side
00710    communication issues, such as 'Connection'.
00711
00712
00713
```

```
00714
00715
00716
00717
00718
00719 Robinson & Coar              Informational                     [Page 19]
00720
00721 RFC 3875                    CGI Version 1.1                 October 2004
00722
00723
00724 4.2.  Request Message-Body
00725
00726    Request data is accessed by the script in a system-defined method;
00727    unless defined otherwise, this will be by reading the 'standard
00728    input' file descriptor or file handle.
00729
00730       Request-Data   = [ request-body ] [ extension-data ]
00731       request-body   = <CONTENT_LENGTH>OCTET
00732       extension-data = *OCTET
00733
00734    A request-body is supplied with the request if the CONTENT_LENGTH is
00735    not NULL.  The server MUST make at least that many bytes available
00736    for the script to read.  The server MAY signal an end-of-file
00737    condition after CONTENT_LENGTH bytes have been read or it MAY supply
00738    extension data.  Therefore, the script MUST NOT attempt to read more
00739    than CONTENT_LENGTH bytes, even if more data is available.  However,
00740    it is not obliged to read any of the data.
00741
00742    For non-parsed header (NPH) scripts (section 5), the server SHOULD
00743    attempt to ensure that the data supplied to the script is precisely
00744    as supplied by the client and is unaltered by the server.
00745
00746    As transfer-codings are not supported on the request-body, the server
00747    MUST remove any such codings from the message-body, and recalculate
00748    the CONTENT_LENGTH.  If this is not possible (for example, because of
00749    large buffering requirements), the server SHOULD reject the client
00750    request.  It MAY also remove content-codings from the message-body.
00751
00752 4.3.  Request Methods
00753
00754    The Request Method, as supplied in the REQUEST_METHOD meta-variable,
00755    identifies the processing method to be applied by the script in
00756    producing a response.  The script author can choose to implement the
00757    methods most appropriate for the particular application.  If the
00758    script receives a request with a method it does not support it SHOULD
00759    reject it with an error (see section 6.3.3).
00760
00761 4.3.1.  GET
00762
00763    The GET method indicates that the script should produce a document
00764    based on the meta-variable values.  By convention, the GET method is
00765    'safe' and 'idempotent' and SHOULD NOT have the significance of
00766    taking an action other than producing a document.
00767
00768    The meaning of the GET method may be modified and refined by
00769    protocol-specific meta-variables.
00770
00771
00772
00773
00774
00775 Robinson & Coar              Informational                     [Page 20]
00776
00777 RFC 3875                    CGI Version 1.1                 October 2004
00778
00779
00780 4.3.2.  POST
00781
00782    The POST method is used to request the script perform processing and
00783    produce a document based on the data in the request message-body, in
00784    addition to meta-variable values.  A common use is form submission in
00785    HTML [18], intended to initiate processing by the script that has a
00786    permanent affect, such a change in a database.
00787
00788    The script MUST check the value of the CONTENT_LENGTH variable before
00789    reading the attached message-body, and SHOULD check the CONTENT_TYPE
00790    value before processing it.
00791
00792 4.3.3.  HEAD
00793
00794    The HEAD method requests the script to do sufficient processing to
00795    return the response header fields, without providing a response
00796    message-body.  The script MUST NOT provide a response message-body
00797    for a HEAD request.  If it does, then the server MUST discard the
00798    message-body when reading the response from the script.
00799
00800 4.3.4.  Protocol-Specific Methods
```

```
00801
00802    The script MAY implement any protocol-specific method, such as
00803    HTTP/1.1 PUT and DELETE; it SHOULD check the value of SERVER_PROTOCOL
00804    when doing so.
00805
00806    The server MAY decide that some methods are not appropriate or
00807    permitted for a script, and may handle the methods itself or return
00808    an error to the client.
00809
00810 6.  CGI Response
00811
00812 6.1.  Response Handling
00813
00814    A script MUST always provide a non-empty response, and so there is a
00815    system-defined method for it to send this data back to the server.
00816    Unless defined otherwise, this will be via the 'standard output' file
00817    descriptor.
00818
00819    The script MUST check the REQUEST_METHOD variable when processing the
00820    request and preparing its response.
00821
00822    The server MAY implement a timeout period within which data must be
00823    received from the script.  If a server implementation defines such a
00824    timeout and receives no data from a script within the timeout period,
00825    the server MAY terminate the script process.
00826
00827 6.2.  Response Types
00828
00829    The response comprises a message-header and a message-body, separated
00830    by a blank line.  The message-header contains one or more header
00831    fields.  The body may be NULL.
00832
00833       generic-response = 1*header-field NL [ response-body ]
00834
00835    The script MUST return one of either a document response, a local
00836    redirect response or a client redirect (with optional document)
00837    response.  In the response definitions below, the order of header
00838    fields in a response is not significant (despite appearing so in the
00839    BNF).  The header fields are defined in section 6.3.
00840
00841       CGI-Response = document-response | local-redir-response |
00842                      client-redir-response | client-redirdoc-response
00843
00844 6.2.1.  Document Response
00845
00846    The CGI script can return a document to the user in a document
00847    response, with an optional error code indicating the success status
00848    of the response.
00849
00850       document-response = Content-Type [ Status ] *other-field NL
00851                           response-body
00852
00853
00854
00855
00856
00857 Robinson & Coar              Informational                    [Page 23]
00858
00859 RFC 3875                    CGI Version 1.1                October 2004
00860
00861
00862    The script MUST return a Content-Type header field.  A Status header
00863    field is optional, and status 200 'OK' is assumed if it is omitted.
00864    The server MUST make any appropriate modifications to the script's
00865    output to ensure that the response to the client complies with the
00866    response protocol version.
00867
00868 6.2.2.  Local Redirect Response
00869
00870    The CGI script can return a URI path and query-string
00871    ('local-pathquery') for a local resource in a Location header field.
00872    This indicates to the server that it should reprocess the request
00873    using the path specified.
00874
00875       local-redir-response = local-Location NL
00876
00877    The script MUST NOT return any other header fields or a message-body,
00878    and the server MUST generate the response that it would have produced
00879    in response to a request containing the URL
00880
00881       scheme "://" server-name ":" server-port local-pathquery
00882
00883 6.2.3.  Client Redirect Response
00884
00885    The CGI script can return an absolute URI path in a Location header
00886    field, to indicate to the client that it should reprocess the request
00887    using the URI specified.
```

```
00888
00889       client-redir-response = client-Location *extension-field NL
00890
00891    The script MUST not provide any other header fields, except for
00892    server-defined CGI extension fields.  For an HTTP client request, the
00893    server MUST generate a 302 'Found' HTTP response message.
00894
00895 6.2.4.  Client Redirect Response with Document
00896
00897    The CGI script can return an absolute URI path in a Location header
00898    field together with an attached document, to indicate to the client
00899    that it should reprocess the request using the URI specified.
00900
00901       client-redirdoc-response = client-Location Status Content-Type
00902                                  *other-field NL response-body
00903
00904    The Status header field MUST be supplied and MUST contain a status
00905    value of 302 'Found', or it MAY contain an extension-code, that is,
00906    another valid status code that means client redirection.  The server
00907    MUST make any appropriate modifications to the script's output to
00908    ensure that the response to the client complies with the response
00909    protocol version.
00910
00911
00912
00913 Robinson & Coar              Informational                     [Page 24]
00914
00915 RFC 3875                    CGI Version 1.1                 October 2004
00916
00917
00918 6.3.  Response Header Fields
00919
00920    The response header fields are either CGI or extension header fields
00921    to be interpreted by the server, or protocol-specific header fields
00922    to be included in the response returned to the client.  At least one
00923    CGI field MUST be supplied; each CGI field MUST NOT appear more than
00924    once in the response.  The response header fields have the syntax:
00925
00926       header-field    = CGI-field | other-field
00927       CGI-field       = Content-Type | Location | Status
00928       other-field     = protocol-field | extension-field
00929       protocol-field  = generic-field
00930       extension-field = generic-field
00931       generic-field   = field-name ":" [ field-value ] NL
00932       field-name      = token
00933       field-value     = *( field-content | LWSP )
00934       field-content   = *( token | separator | quoted-string )
00935
00936    The field-name is not case sensitive.  A NULL field value is
00937    equivalent to a field not being sent.  Note that each header field in
00938    a CGI-Response MUST be specified on a single line; CGI/1.1 does not
00939    support continuation lines.  Whitespace is permitted between the ":"
00940    and the field-value (but not between the field-name and the ":"), and
00941    also between tokens in the field-value.
00942
00943 6.3.1.  Content-Type
00944
00945    The Content-Type response field sets the Internet Media Type [6] of
00946    the entity body.
00947
00948       Content-Type = "Content-Type:" media-type NL
00949
00950    If an entity body is returned, the script MUST supply a Content-Type
00951    field in the response.  If it fails to do so, the server SHOULD NOT
00952    attempt to determine the correct content type.  The value SHOULD be
00953    sent unmodified to the client, except for any charset parameter
00954    changes.
00955
00956    Unless it is otherwise system-defined, the default charset assumed by
00957    the client for text media-types is ISO-8859-1 if the protocol is HTTP
00958    and US-ASCII otherwise.  Hence the script SHOULD include a charset
00959    parameter.  See section 3.4.1 of the HTTP/1.1 specification [4] for a
00960    discussion of this issue.
00961
00962
00963
00964
00965
00966
00967
00968
00969 Robinson & Coar              Informational                     [Page 25]
00970
00971 RFC 3875                    CGI Version 1.1                 October 2004
00972
00973
00974 6.3.2.  Location
```

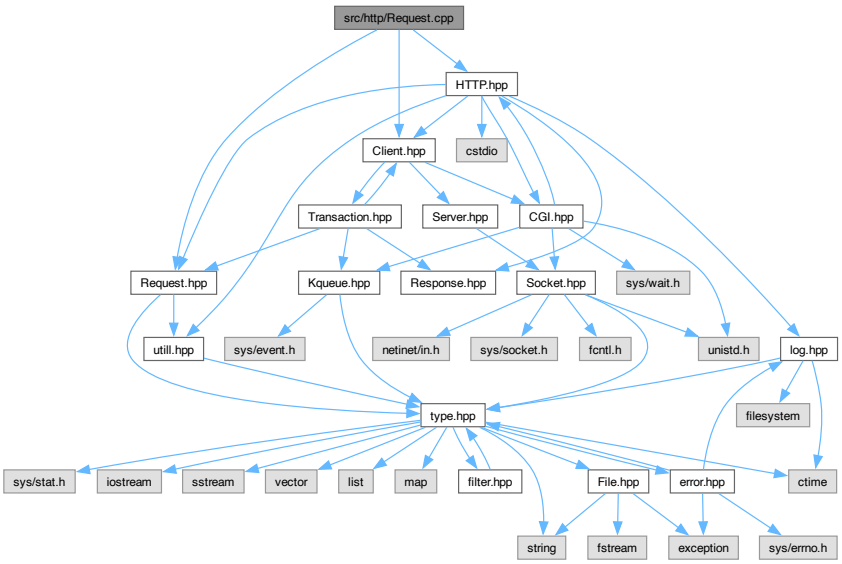
```
00975
00976    The Location header field is used to specify to the server that the
00977    script is returning a reference to a document rather than an actual
00978    document (see sections 6.2.3 and 6.2.4).  It is either an absolute
00979    URI (optionally with a fragment identifier), indicating that the
00980    client is to fetch the referenced document, or a local URI path
00981    (optionally with a query string), indicating that the server is to
00982    fetch the referenced document and return it to the client as the
00983    response.
00984
00985       Location        = local-Location | client-Location
00986       client-Location = "Location:" fragment-URI NL
00987       local-Location  = "Location:" local-pathquery NL
00988       fragment-URI    = absoluteURI [ "#" fragment ]
00989       fragment        = *uric
00990       local-pathquery = abs-path [ "?" query-string ]
00991       abs-path        = "/" path-segments
00992       path-segments   = segment *( "/" segment )
00993       segment         = *pchar
00994       pchar           = unreserved | escaped | extra
00995       extra           = ":" | "@" | "&" | "=" | "+" | "$" | ","
00996
00997    The syntax of an absoluteURI is incorporated into this document from
00998    that specified in RFC 2396 [2] and RFC 2732 [7].  A valid absoluteURI
00999    always starts with the name of scheme followed by ":"; scheme names
01000    start with a letter and continue with alphanumerics, "+", "-" or ".".
01001    The local URI path and query must be an absolute path, and not a
01002    relative path or NULL, and hence must start with a "/".
01003
01004    Note that any message-body attached to the request (such as for a
01005    POST request) may not be available to the resource that is the target
01006    of the redirect.
01007
01008 6.3.3.  Status
01009
01010    The Status header field contains a 3-digit integer result code that
01011    indicates the level of success of the script's attempt to handle the
01012    request.
01013
01014       Status         = "Status:" status-code SP reason-phrase NL
01015       status-code    = "200" | "302" | "400" | "501" | extension-code
01016       extension-code = 3digit
01017       reason-phrase  = *TEXT
01018
01019    Status code 200 'OK' indicates success, and is the default value
01020    assumed for a document response.  Status code 302 'Found' is used
01021    with a Location header field and response message-body.  Status code
01022
01023
01024
01025 Robinson & Coar              Informational                    [Page 26]
01026
01027 RFC 3875                    CGI Version 1.1                October 2004
01028
01029
01030    400 'Bad Request' may be used for an unknown request format, such as
01031    a missing CONTENT_TYPE.  Status code 501 'Not Implemented' may be
01032    returned by a script if it receives an unsupported REQUEST_METHOD.
01033
01034    Other valid status codes are listed in section 6.1.1 of the HTTP
01035    specifications [1], [4], and also the IANA HTTP Status Code Registry
01036    [8] and MAY be used in addition to or instead of the ones listed
01037    above.  The script SHOULD check the value of SERVER_PROTOCOL before
01038    using HTTP/1.1 status codes.  The script MAY reject with error 405
01039    'Method Not Allowed' HTTP/1.1 requests made using a method it does
01040    not support.
01041
01042    Note that returning an error status code does not have to mean an
01043    error condition with the script itself.  For example, a script that
01044    is invoked as an error handler by the server should return the code
01045    appropriate to the server's error condition.
01046
01047    The reason-phrase is a textual description of the error to be
01048    returned to the client for human consumption.
01049
01050 6.3.4.  Protocol-Specific Header Fields
01051
01052    The script MAY return any other header fields that relate to the
01053    response message defined by the specification for the SERVER_PROTOCOL
01054    (HTTP/1.0 [1] or HTTP/1.1 [4]).  The server MUST translate the header
01055    data from the CGI header syntax to the HTTP header syntax if these
01056    differ.  For example, the character sequence for newline (such as
01057    UNIX's US-ASCII LF) used by CGI scripts may not be the same as that
01058    used by HTTP (US-ASCII CR followed by LF).
01059
01060    The script MUST NOT return any header fields that relate to
01061    client-side communication issues and could affect the server's
```

```
01062    ability to send the response to the client.  The server MAY remove
01063    any such header fields returned by the client.  It SHOULD resolve any
01064    conflicts between header fields returned by the script and header
01065    fields that it would otherwise send itself.
01066
01067 6.3.5.  Extension Header Fields
01068
01069    There may be additional implementation-defined CGI header fields,
01070    whose field names SHOULD begin with "X-CGI-".  The server MAY ignore
01071    (and delete) any unrecognised header fields with names beginning "X-
01072    CGI-" that are received from the script.
01073
01074
01075
01076
01077
01078
01079
01080
01081 Robinson & Coar              Informational                    [Page 27]
01082
01083 RFC 3875                   CGI Version 1.1                October 2004
01084
01085
01086 6.4.  Response Message-Body
01087
01088    The response message-body is an attached document to be returned to
01089    the client by the server.  The server MUST read all the data provided
01090    by the script, until the script signals the end of the message-body
01091    by way of an end-of-file condition.  The message-body SHOULD be sent
01092    unmodified to the client, except for HEAD requests or any required
01093    transfer-codings, content-codings or charset conversions.
01094
01095       response-body = *OCTET
01096
01097 */
```

7.42 src/http/Request.cpp File Reference

```
#include "HTTP.hpp"
#include "Client.hpp"
#include "Request.hpp"
```

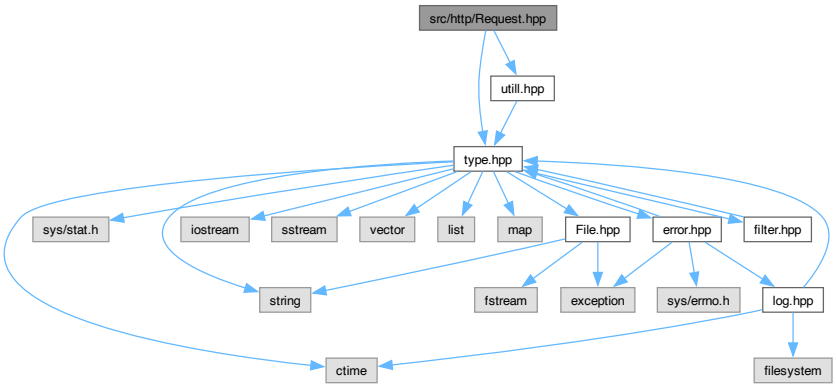
Include dependency graph for Request.cpp:



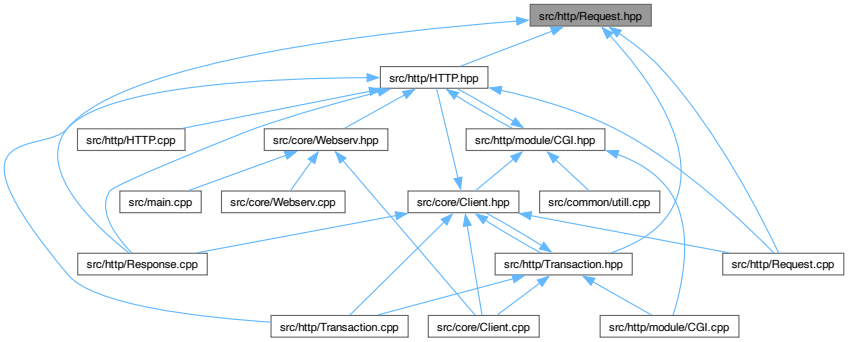
7.43 src/http/Request.hpp File Reference

```
#include "type.hpp"
#include "utill.hpp"
```

Include dependency graph for Request.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Request

7.44 Request.hpp

Go to the documentation of this file.

```
00001 #ifndef REQUEST_HPP
00002 # define REQUEST_HPP
00003
00004 # include "type.hpp"
00005 # include "utill.hpp"
00006
00007 class Client;
00008
```

```
00009 class Request {
00010     public:
00011         Request( const Client& );
00012         ~Request( void );
00013 
00014         fstat_t                     info;
00015 
00016         const Client&             client( void ) const;
00017         const config_t&             config( void ) const;
00018         const location_t&         location( void ) const;
00019 
00020         const request_line_t&     line( void ) const;
00021         const request_header_t&     header( void ) const;
00022         const sstream_t&           body( void ) const;
00023 
00024     private:
00025         const Client&             _client;
00026         size_t                      _config;
00027         size_t                      _location;
00028 
00029         request_line_t                _line;
00030         request_header_t            _header;
00031 
00032         void                        _parse( const sstream_t& );
00033 
00034         void                        _parseLine( const str_t& );
00035         void                        _assignMethod( str_t );
00036         void                        _assignURI( str_t );
00037         void                        _assignVersion( str_t );
00038 
00039         void                        _parseHeader( const str_t& );
00040         ssize_t                     _add( vec_uint_t&, ssize_t );
00041 
00042         str_t                      _token( isstream_t&, const char& );
00043 
00044         void                        _valid( void );
00045         void                        _redirectURI( void );
00046 
00047 };
00048 
00049 #endif
00050 
00051 /*
00052 6. Message Body
00053 The message body (if any) of an HTTP/1.1 message is used to carry
00054 content (Section 6.4 of [HTTP]) for the request or response. The message
00055 body is identical to the content unless a transfer coding has been applied,
00056 as described in Section 6.1.
00057 
00058   message-body = *OCTET
00059 The rules for determining when a message body is present in an HTTP/1.1
00060 message differ for requests and responses.
00061 
00062 The presence of a message body in a request is signaled by a Content-Length
00063 or Transfer-Encoding header field. Request message framing is independent of method semantics.
00064 
00065 The presence of a message body in a response, as detailed in Section 6.3, depends
00066 on both the request method to which it is responding and the response status code.
00067 This corresponds to when response content is allowed by HTTP semantics (Section 6.4.1 of [HTTP]).
00068 
00069 6.1. Transfer-Encoding
00070 The Transfer-Encoding header field lists the transfer coding names corresponding
00071 to the sequence of transfer codings that have been (or will be) applied to the
00072 content in order to form the message body. Transfer codings are defined in Section 7.
00073 
00074   Transfer-Encoding = #transfer-coding
00075                       ; defined in [HTTP], Section 10.1.4
00076 Transfer-Encoding is analogous to the Content-Transfer-Encoding field of MIME, which
00077 was designed to enable safe transport of binary data over a 7-bit transport service
00078 ([RFC2045], Section 6). However, safe transport has a different focus for an 8bit-clean transfer
      protocol.
00079 In HTTP's case, Transfer-Encoding is primarily intended to accurately delimit dynamically generated
      content.
00080 It also serves to distinguish encodings that are only applied in transit from the encodings that are
00081 a characteristic of the selected representation.
00082 
00083 A recipient MUST be able to parse the chunked transfer coding (Section 7.1) because
00084 it plays a crucial role in framing messages when the content size is not known in advance.
00085 A sender MUST NOT apply the chunked transfer coding more than once to a message body
00086 (i.e., chunking an already chunked message is not allowed). If any transfer coding other
00087 than chunked is applied to a request's content, the sender MUST apply chunked as the final
00088 transfer coding to ensure that the message is properly framed. If any transfer coding other
00089 than chunked is applied to a response's content, the sender MUST either apply chunked as the
00090 final transfer coding or terminate the message by closing the connection.
00091 
00092 For example,
00093 
```

```
00094 Transfer-Encoding: gzip, chunked
00095 indicates that the content has been compressed using the gzip coding and then
00096 chunked using the chunked coding while forming the message body.
00097
00098 Unlike Content-Encoding (Section 8.4.1 of [HTTP]), Transfer-Encoding is a property of the message,
00099 not of the representation. Any recipient along the request/response chain MAY decode the received
00100 transfer coding(s) or apply additional transfer coding(s) to the message body, assuming that
      corresponding
00101 changes are made to the Transfer-Encoding field value. Additional information about the encoding
00102 parameters can be provided by other header fields not defined by this specification.
00103
00104 Transfer-Encoding MAY be sent in a response to a HEAD request or in a 304 (Not Modified) response
00105 (Section 15.4.5 of [HTTP]) to a GET request, neither of which includes a message body, to indicate
00106 that the origin server would have applied a transfer coding to the message body if the request had
      been an
00107 unconditional GET. This indication is not required, however, because any recipient on the response
      chain
00108 (including the origin server) can remove transfer codings when they are not needed.
00109
00110 A server MUST NOT send a Transfer-Encoding header field in any response with a status code of 1xx
      (Informational)
00111 or 204 (No Content). A server MUST NOT send a Transfer-Encoding header field in any 2xx (Successful)
00112 response to a CONNECT request (Section 9.3.6 of [HTTP]).
00113
00114 A server that receives a request message with a transfer coding it does not understand SHOULD
00115 respond with 501 (Not Implemented).
00116
00117 Transfer-Encoding was added in HTTP/1.1. It is generally assumed that implementations advertising
00118 only HTTP/1.0 support will not understand how to process transfer-encoded content, and that an
      HTTP/1.0
00119 message received with a Transfer-Encoding is likely to have been forwarded without proper handling of
00120 the chunked transfer coding in transit.
00121
00122 A client MUST NOT send a request containing Transfer-Encoding unless it knows the server will handle
00123 HTTP/1.1 requests (or later minor revisions); such knowledge might be in the form of specific user
00124 configuration or by remembering the version of a prior received response. A server MUST NOT send a
00125 response containing Transfer-Encoding unless the corresponding request indicates HTTP/1.1 (or later
      minor revisions).
00126
00127 Early implementations of Transfer-Encoding would occasionally send both a chunked transfer coding for
00128 message framing and an estimated Content-Length header field for use by progress bars. This is why
00129 Transfer-Encoding is defined as overriding Content-Length, as opposed to them being mutually
      incompatible.
00130 Unfortunately, forwarding such a message can lead to vulnerabilities regarding request smuggling
      (Section 11.2)
00131 or response splitting (Section 11.1) attacks if any downstream recipient fails to parse the message
00132 according to this specification, particularly when a downstream recipient only implements HTTP/1.0.
00133
00134 A server MAY reject a request that contains both Content-Length and Transfer-Encoding or process
00135 such a request in accordance with the Transfer-Encoding alone. Regardless, the server MUST close
00136 the connection after responding to such a request to avoid the potential attacks.
00137
00138 A server or client that receives an HTTP/1.0 message containing a Transfer-Encoding header field
00139 MUST treat the message as if the framing is faulty, even if a Content-Length is present, and close
00140 the connection after processing the message. The message sender might have retained a portion of
00141 the message, in buffer, that could be misinterpreted by further use of the connection.
00142
00143 6.2. Content-Length
00144 When a message does not have a Transfer-Encoding header field, a Content-Length header field
00145 (Section 8.6 of [HTTP]) can provide the anticipated size, as a decimal number of octets, for
00146 potential content. For messages that do include content, the Content-Length field value provides
00147 the framing information necessary for determining where the data (and message) ends. For messages
00148 that do not include content, the Content-Length indicates the size of the selected representation
00149 (Section 8.6 of [HTTP]).
00150
00151 A sender MUST NOT send a Content-Length header field in any message that contains a Transfer-Encoding
      header field.
00152
00153 Note: HTTP's use of Content-Length for message framing differs significantly from the same field's
00154 use in MIME, where it is an optional field used only within the "message/external-body" media-type.
00155
00156 6.3. Message Body Length
00157 The length of a message body is determined by one of the following (in order of precedence):
00158
00159 Any response to a HEAD request and any response with a 1xx (Informational), 204 (No Content), or
00160 304 (Not Modified) status code is always terminated by the first empty line after the header fields,
00161 regardless of the header fields present in the message, and thus cannot contain a message body or
      trailer section.
00162
00163 Any 2xx (Successful) response to a CONNECT request implies that the connection will become a tunnel
00164 immediately after the empty line that concludes the header fields. A client MUST ignore any
      Content-Length
00165 or Transfer-Encoding header fields received in such a message.
00166
00167 If a message is received with both a Transfer-Encoding and a Content-Length header field,
00168 the Transfer-Encoding overrides the Content-Length. Such a message might indicate an attempt to
00169 perform request smuggling (Section 11.2) or response splitting (Section 11.1) and ought to be handled
```

```
      as an error.
00170 An intermediary that chooses to forward the message MUST first remove the received Content-Length
      field and process
00171 the Transfer-Encoding (as described below) prior to forwarding the message downstream.
00172
00173 If a Transfer-Encoding header field is present and the chunked transfer coding (Section 7.1) is the
      final encoding,
00174 the message body length is determined by reading and decoding the chunked data until the transfer
      coding indicates
00175 the data is complete.
00176
00177 If a Transfer-Encoding header field is present in a response and the chunked transfer coding is not
      the final encoding,
00178 the message body length is determined by reading the connection until it is closed by the server.
00179
00180 If a Transfer-Encoding header field is present in a request and the chunked transfer coding is not the
      final encoding,
00181 the message body length cannot be determined reliably; the server MUST respond with the 400 (Bad
      Request) status code and then close the connection.
00182
00183 If a message is received without Transfer-Encoding and with an invalid Content-Length header field,
      then the message
00184 framing is invalid and the recipient MUST treat it as an unrecoverable error, unless the field value
      can be successfully
00185 parsed as a comma-separated list (Section 5.6.1 of [HTTP]), all values in the list are valid, and all
      values in the list
00186 are the same (in which case, the message is processed with that single value used as the
      Content-Length field value).
00187 If the unrecoverable error is in a request message, the server MUST respond with a 400 (Bad Request)
      status code and
00188 then close the connection. If it is in a response message received by a proxy, the proxy MUST close
      the connection to the server,
00189 discard the received response, and send a 502 (Bad Gateway) response to the client. If it is in a
      response message received by
00190 a user agent, the user agent MUST close the connection to the server and discard the received
      response.
00191
00192 If a valid Content-Length header field is present without Transfer-Encoding, its decimal value defines
      the expected
00193 message body length in octets. If the sender closes the connection or the recipient times out before
      the indicated
00194 number of octets are received, the recipient MUST consider the message to be incomplete and close the
      connection.
00195
00196 If this is a request message and none of the above are true, then the message body length is zero (no
      message body is present).
00197 Otherwise, this is a response message without a declared message body length, so the message body
      length
00198 is determined by the number of octets received prior to the server closing the connection.
00199 Since there is no way to distinguish a successfully completed, close-delimited response message from a
      partially
00200 received message interrupted by network failure, a server SHOULD generate encoding or length-delimited
      messages
00201 whenever possible. The close-delimiting feature exists primarily for backwards compatibility with
      HTTP/1.0.
00202
00203 Note: Request messages are never close-delimited because they are always explicitly framed by
00204 length or transfer coding, with the absence of both implying the request ends immediately after the
      header section.
00205
00206 A server MAY reject a request that contains a message body but not a Content-Length by responding with
      411 (Length Required).
00207
00208 Unless a transfer coding other than chunked has been applied, a client that sends a request
00209 containing a message body SHOULD use a valid Content-Length header field if the message body
00210 length is known in advance, rather than the chunked transfer coding, since some existing services
00211 respond to chunked with a 411 (Length Required) status code even though they understand the
00212 chunked transfer coding. This is typically because such services are implemented via a gateway
00213 that requires a content length in advance of being called, and the server is unable or unwilling
00214 to buffer the entire request before processing.
00215
00216 A user agent that sends a request that contains a message body MUST send either a valid Content-Length
00217 header field or use the chunked transfer coding. A client MUST NOT use the chunked transfer coding
      unless
00218 it knows the server will handle HTTP/1.1 (or later) requests; such knowledge can be in the form of
00219 specific user configuration or by remembering the version of a prior received response.
00220
00221 If the final response to the last request on a connection has been completely received and there
00222 remains additional data to read, a user agent MAY discard the remaining data or attempt to determine
00223 if that data belongs as part of the prior message body, which might be the case if the prior
00224 message's Content-Length value is incorrect. A client MUST NOT process, cache, or forward such extra
00225 data as a separate response, since such behavior would be vulnerable to cache poisoning.
00226
00227 7. Transfer Codings
00228 Transfer coding names are used to indicate an encoding transformation that has been, can be, or might
00229 need to be applied to a message's content in order to ensure "safe transport" through the network.
00230 This differs from a content coding in that the transfer coding is a property of the message rather
```

```
00231 than a property of the representation that is being transferred.
00232
00233 All transfer-coding names are case-insensitive and ought to be registered within the HTTP Transfer
00234 Coding registry, as defined in Section 7.3. They are used in the Transfer-Encoding (Section 6.1)
00235 and TE (Section 10.1.4 of [HTTP]) header fields (the latter also defining the "transfer-coding"
      grammar).
00236
00237 7.1. Chunked Transfer Coding
00238 The chunked transfer coding wraps content in order to transfer it as a series of chunks,
00239 each with its own size indicator, followed by an OPTIONAL trailer section containing trailer fields.
00240 Chunked enables content streams of unknown size to be transferred as a sequence of length-delimited
      buffers,
00241 which enables the sender to retain connection persistence and the recipient to know when
00242 it has received the entire message.
00243
00244   chunked-body   = *chunk
00245                    last-chunk
00246                    trailer-section
00247                    CRLF
00248
00249   chunk          = chunk-size [ chunk-ext ] CRLF
00250                    chunk-data CRLF
00251   chunk-size     = 1*HEXDIG
00252   last-chunk     = 1*("0") [ chunk-ext ] CRLF
00253
00254   chunk-data     = 1*OCTET ; a sequence of chunk-size octets
00255
00256 The chunk-size field is a string of hex digits indicating the size of the chunk-data in octets.
00257 The chunked transfer coding is complete when a chunk with a chunk-size of zero is received,
00258 possibly followed by a trailer section, and finally terminated by an empty line.
00259
00260 A recipient MUST be able to parse and decode the chunked transfer coding.
00261
00262 HTTP/1.1 does not define any means to limit the size of a chunked response such that an intermediary
00263 can be assured of buffering the entire response. Additionally, very large chunk sizes may cause
00264 overflows or loss of precision if their values are not represented accurately in a receiving
      implementation.
00265 Therefore, recipients MUST anticipate potentially large hexadecimal numerals and prevent parsing
00266 errors due to integer conversion overflows or precision loss due to integer representation.
00267
00268 The chunked coding does not define any parameters. Their presence SHOULD be treated as an error.
00269
00270 7.1.1. Chunk Extensions
00271 The chunked coding allows each chunk to include zero or more chunk extensions, immediately
00272 following the chunk-size, for the sake of supplying per-chunk metadata (such as a signature or hash),
00273 mid-message control information, or randomization of message body size.
00274
00275   chunk-ext      = *( BWS ";" BWS chunk-ext-name
00276                       [ BWS "=" BWS chunk-ext-val ] )
00277
00278   chunk-ext-name = token
00279   chunk-ext-val  = token / quoted-string
00280 The chunked coding is specific to each connection and is likely to be removed or recoded by
00281 each recipient (including intermediaries) before any higher-level application would have a
00282 chance to inspect the extensions. Hence, the use of chunk extensions is generally limited to
00283 specialized HTTP services such as "long polling" (where client and server can have shared
00284 expectations regarding the use of chunk extensions) or for padding within an end-to-end
00285 secured connection.
00286
00287 A recipient MUST ignore unrecognized chunk extensions. A server ought to limit the total
00288 length of chunk extensions received in a request to an amount reasonable for the services provided,
00289 in the same way that it applies length limitations and timeouts for other parts of a message,
00290 and generate an appropriate 4xx (Client Error) response if that amount is exceeded.
00291
00292 7.1.2. Chunked Trailer Section
00293 A trailer section allows the sender to include additional fields at the end of a chunked
00294 message in order to supply metadata that might be dynamically generated while the content is
00295 sent, such as a message integrity check, digital signature, or post-processing status.
00296 The proper use and limitations of trailer fields are defined in Section 6.5 of [HTTP].
00297
00298   trailer-section   = *( field-line CRLF )
00299 A recipient that removes the chunked coding from a message MAY selectively retain or
00300 discard the received trailer fields. A recipient that retains a received trailer field
00301 MUST either store/forward the trailer field separately from the received header fields
00302 or merge the received trailer field into the header section. A recipient MUST NOT merge
00303 a received trailer field into the header section unless its corresponding header field
00304 definition explicitly permits and instructs how the trailer field value can be safely merged.
00305
00306 7.1.3. Decoding Chunked
00307 A process for decoding the chunked transfer coding can be represented in pseudo-code as:
00308
00309   length := 0
00310   read chunk-size, chunk-ext (if any), and CRLF
00311   while (chunk-size > 0) {
00312      read chunk-data and CRLF
00313      append chunk-data to content
00314      length := length + chunk-size
```

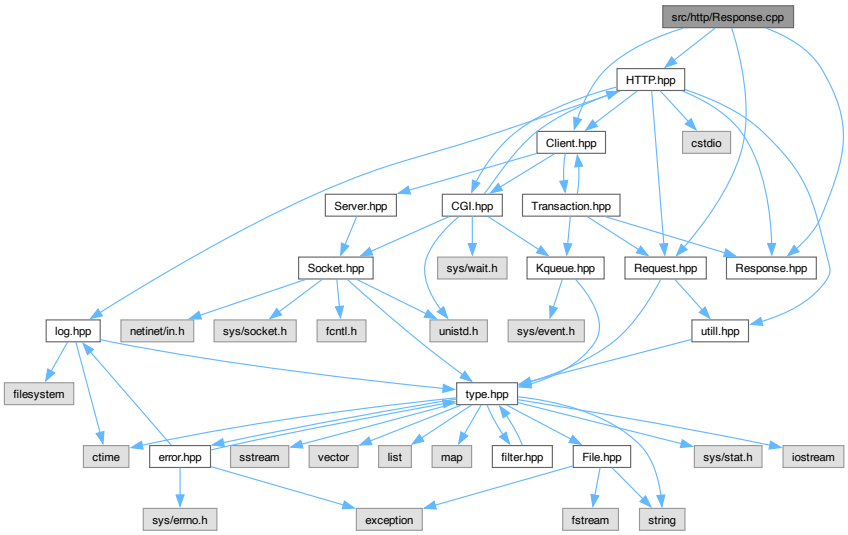
```
00315       read chunk-size, chunk-ext (if any), and CRLF
00316    }
00317    read trailer field
00318    while (trailer field is not empty) {
00319       if (trailer fields are stored/forwarded separately) {
00320           append trailer field to existing trailer fields
00321       }
00322       else if (trailer field is understood and defined as mergeable) {
00323           merge trailer field with existing header fields
00324       }
00325       else {
00326           discard trailer field
00327       }
00328       read trailer field
00329    }
00330    Content-Length := length
00331    Remove "chunked" from Transfer-Encoding
00332 7.2. Transfer Codings for Compression
00333 The following transfer coding names for compression are defined by the same
00334 algorithm as their corresponding content coding:
00335
00336 compress (and x-compress)
00337 See Section 8.4.1.1 of [HTTP].
00338 deflate
00339 See Section 8.4.1.2 of [HTTP].
00340 gzip (and x-gzip)
00341 See Section 8.4.1.3 of [HTTP].
00342 The compression codings do not define any parameters. The presence of parameters with
00343 any of these compression codings SHOULD be treated as an error.
00344
00345 7.3. Transfer Coding Registry
00346 The "HTTP Transfer Coding Registry" defines the namespace for transfer coding names.
00347 It is maintained at <https://www.iana.org/assignments/http-parameters>.
00348
00349 Registrations MUST include the following fields:
00350
00351 Name
00352 Description
00353 Pointer to specification text
00354 Names of transfer codings MUST NOT overlap with names of content codings
00355 (Section 8.4.1 of [HTTP]) unless the encoding transformation is identical,
00356 as is the case for the compression codings defined in Section 7.2.
00357
00358 The TE header field (Section 10.1.4 of [HTTP]) uses a pseudo-parameter named "q"
00359 as the rank value when multiple transfer codings are acceptable. Future registrations
00360 of transfer codings SHOULD NOT define parameters called "q" (case-insensitively)
00361 in order to avoid ambiguities.
00362
00363 Values to be added to this namespace require IETF Review (see Section 4.8 of [RFC8126])
00364 and MUST conform to the purpose of transfer coding defined in this specification.
00365
00366 Use of program names for the identification of encoding formats is not desirable
00367 and is discouraged for future encodings.
00368
00369 7.4. Negotiating Transfer Codings
00370 The TE field (Section 10.1.4 of [HTTP]) is used in HTTP/1.1 to indicate what
00371 transfer codings, besides chunked, the client is willing to accept in the response
00372 and whether the client is willing to preserve trailer fields in a chunked transfer coding.
00373
00374 A client MUST NOT send the chunked transfer coding name in TE; chunked is always
00375 acceptable for HTTP/1.1 recipients.
00376
00377 Three examples of TE use are below.
00378
00379 TE: deflate
00380 TE:
00381 TE: trailers, deflate;q=0.5
00382 When multiple transfer codings are acceptable, the client MAY rank the codings by
00383 preference using a case-insensitive "q" parameter (similar to the qvalues used in
00384 content negotiation fields; see Section 12.4.2 of [HTTP]). The rank value is a real
00385 number in the range 0 through 1, where 0.001 is the least preferred and 1 is the
00386 most preferred; a value of 0 means "not acceptable".
00387
00388 If the TE field value is empty or if no TE field is present, the only acceptable
00389 transfer coding is chunked. A message with no transfer coding is always acceptable.
00390
00391 The keyword "trailers" indicates that the sender will not discard trailer fields,
00392 as described in Section 6.5 of [HTTP].
00393
00394 Since the TE header field only applies to the immediate connection, a sender of TE
00395 MUST also send a "TE" connection option within the Connection header field (Section 7.6.1 of [HTTP])
00396 in order to prevent the TE header field from being forwarded by
00397 intermediaries that do not support its semantics.
00398
00399 8. Handling Incomplete Messages
00400 A server that receives an incomplete request message, usually due to a canceled
00401 request or a triggered timeout exception, MAY send an error response prior to
```

```
00402 closing the connection.
00403
00404 A client that receives an incomplete response message, which can occur when a
00405 connection is closed prematurely or when decoding a supposedly chunked transfer
00406 coding fails, MUST record the message as incomplete. Cache requirements for
00407 incomplete responses are defined in Section 3.3 of [CACHING].
00408
00409 If a response terminates in the middle of the header section (before the empty
00410 line is received) and the status code might rely on header fields to convey the
00411 full meaning of the response, then the client cannot assume that meaning has been
00412 conveyed; the client might need to repeat the request in order to determine what action to take next.
00413
00414 A message body that uses the chunked transfer coding is incomplete if the zero-sized
00415 chunk that terminates the encoding has not been received. A message that uses a valid
00416 Content-Length is incomplete if the size of the message body received (in octets) is
00417 less than the value given by Content-Length. A response that has neither chunked transfer
00418 coding nor Content-Length is terminated by closure of the connection and, if the header
00419 section was received intact, is considered complete unless an error was indicated
00420 by the underlying connection (e.g., an "incomplete close" in TLS would leave
00421 the response incomplete, as described in Section 9.8).
00422 */
```

7.45 src/http/Response.cpp File Reference

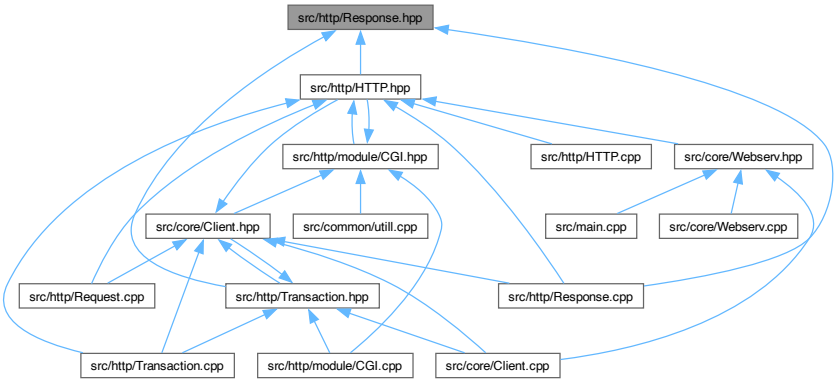
```
#include "HTTP.hpp"
#include "Client.hpp"
#include "Request.hpp"
#include "Response.hpp"
```

Include dependency graph for Response.cpp:



7.46 src/http/Response.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Response

7.47 Response.hpp

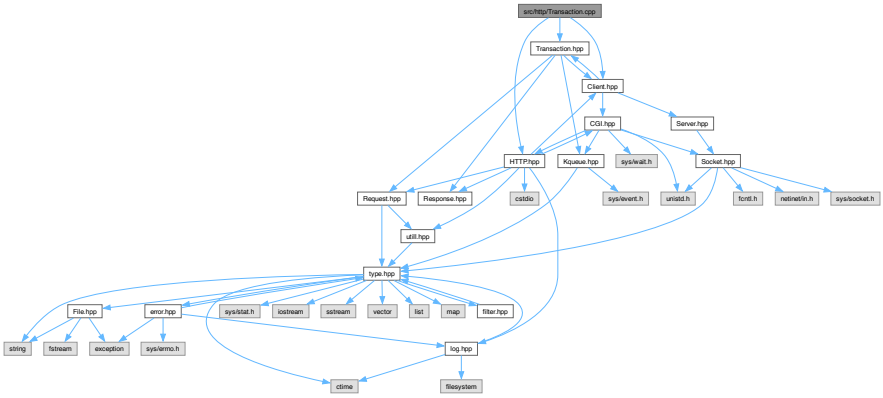
Go to the documentation of this file.

```
00001 #ifndef RESPONSE_HPP
00002 # define RESPONSE_HPP
00003
00004 class Client;
00005 class Request;
00006
00007 class Response {
00008     public:
00009         Response( void );
00010         Response( const uint_t&, const config_t& );
00011         ~Response( void );
00012
00013         const response_line_t&       line( void ) const;
00014         const response_header_t&   header( void ) const ;
00015         const sstream_t&           body( void ) const;
00016
00017         void                        act( const Request& );
00018
00019     private:
00020         response_line_t             _line;
00021         response_header_t          _header;
00022         sstream_t                  _body;
00023
00024         void                        _doMethod( const Request& );
00025         void                        _doMethodValid( const Request& );
00026         void                        _addServerInfo( const connection_e& );
00027
00028         void                        _index( const Request& );
00029         void                        _indexFile( const Request&, const path_t&, const fstat_t& );
00030         path_t                        _indexFileValid( const Request&, fstat_t& );
00031         void                        _indexAuto( const Request& );
00032         void                        _indexAutoBuild( const Request& );
00033         path_t                        _indexURIConceal( const Request&, const path_t& );
00034
00035         void                        _redirect( const path_t&, const uint_t& );
00036
00037         void                        _errpage( const uint_t&, const config_t& );
00038         void                        _errpageBuild( const uint_t& );
00039         void                        _mime( const str_t& );
00040 };
00041
00042 #endif
```

7.48 src/http/Transaction.cpp File Reference

```
#include "Transaction.hpp"
#include "HTTP.hpp"
#include "Client.hpp"
```

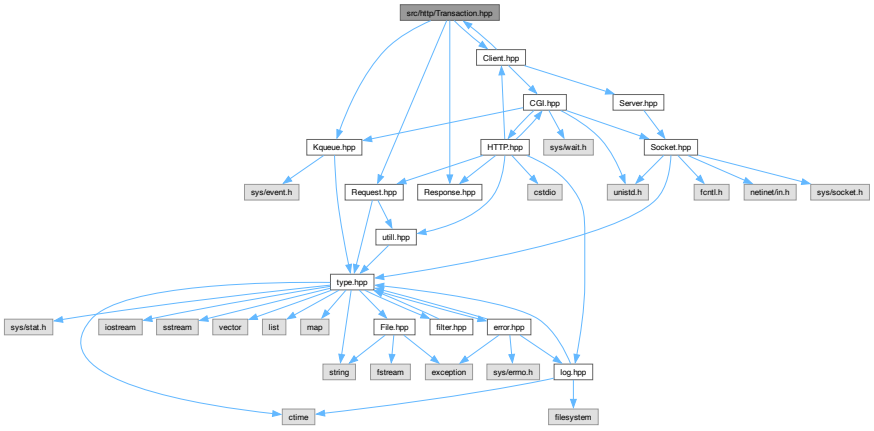
Include dependency graph for Transaction.cpp:



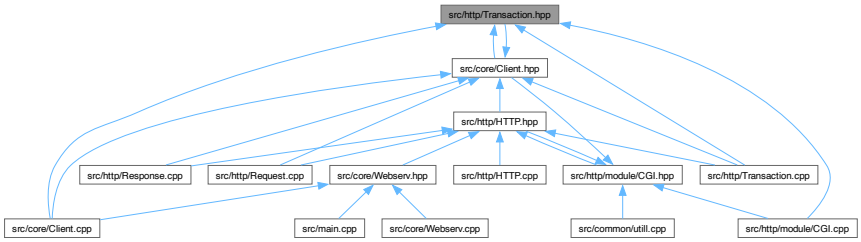
7.49 src/http/Transaction.hpp File Reference

```
#include "Kqueue.hpp"
#include "Request.hpp"
#include "Response.hpp"
#include "Client.hpp"
```

Include dependency graph for Transaction.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct message_s
- class Transaction

Macros

- #define SIZE_BUFF 1024
- #define SIZE_BUFF_RECV 2048

Typedefs

- typedef struct message_s message_t

7.49.1 Macro Definition Documentation

7.49.1.1 SIZE_BUFF

```
#define SIZE_BUFF 1024
```

7.49.1.2 SIZE_BUFF_RECV

```
#define SIZE_BUFF_RECV 2048
```

7.49.2 Typedef Documentation

7.49.2.1 message_t

```
typedef struct message_s message_t
```

7.50 Transaction.hpp

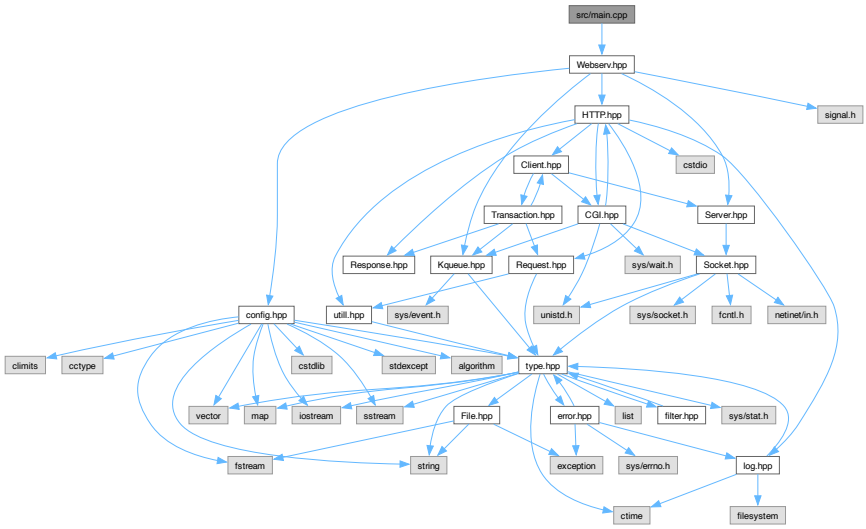
Go to the documentation of this file.

```
00001 #ifndef TRANSACTION_HPP
00002 # define TRANSACTION_HPP
00003
00004 # include "Kqueue.hpp"
00005 # include "Request.hpp"
00006 # include "Response.hpp"
00007
00008 # define SIZE_BUFF      1024
00009 # define SIZE_BUFF_RECV 2048
00010
00011 typedef struct message_s{
00012     message_s();
00013
00014     void    reset();
00015
00016     sstream_t  head;
00017     ssize_t     head_read;
00018     bool        head_done;
00019
00020     sstream_t  body;
00021     ssize_t     body_read;
00022     ssize_t     body_size;
00023
00024     bool        chunk;
00025     size_t      incomplete;
00026 }   message_t;
00027
00028 static const char hexdigit[17] = "0123456789ABCDEF";
00029
00030 class Transaction {
00031     public:
00032         Transaction( Client&, Kqueue& );
00033
00034         const config_t&     config( void );
00035         connection_e        connection( void );
00036
00037         static bool         takeHead( message_t&, char*, ssize_t& );
00038         static bool         takeBody( message_t&, const process_t&, const char*, const ssize_t& );
00039
00040         static void         build( const Response&, message_t& );
00041         static void         buildError( const uint_t&, Client& );
00042
00043         void                act( void );
00044
00045     private:
00046         Client&               _cl;
00047
00048         Request              _rqst;
00049         Response            _rspn;
00050
00051         static bool         _recvBodyPlain( message_t&, const process_t&, const char*, const ssize_t&
      );
00052         static bool         _recvBodyChunk( message_t&, const process_t&, const char* );
00053         static bool         _recvBodyChunkData( message_t&, const process_t&, isstream_t& );
00054         static bool         _recvBodyChunkPredata( message_t&, const process_t& );
00055         static bool         _recvBodyChunkIncomplete( message_t&, const process_t&, isstream_t& );
00056
00057         static void         _buildLine( const Response&, sstream_t& );
00058         static void         _buildHeader( const Response&, sstream_t& );
00059         static void         _buildHeaderName( uint_t, sstream_t& );
00060         static void         _buildHeaderValue( const response_header_t&, uint_t, sstream_t& );
00061         static void         _buildBody( const Response&, message_t& );
00062
00063         void                _validRequest( void );
00064         void                _setBodyEnd( void );
00065         bool                _invokeCGI( const Request&, process_t& );
00066 };
00067
00068 # include "Client.hpp"
00069
00070 #endif
```

7.51 src/main.cpp File Reference

```
#include "Webserv.hpp"
```

Include dependency graph for main.cpp:



Functions

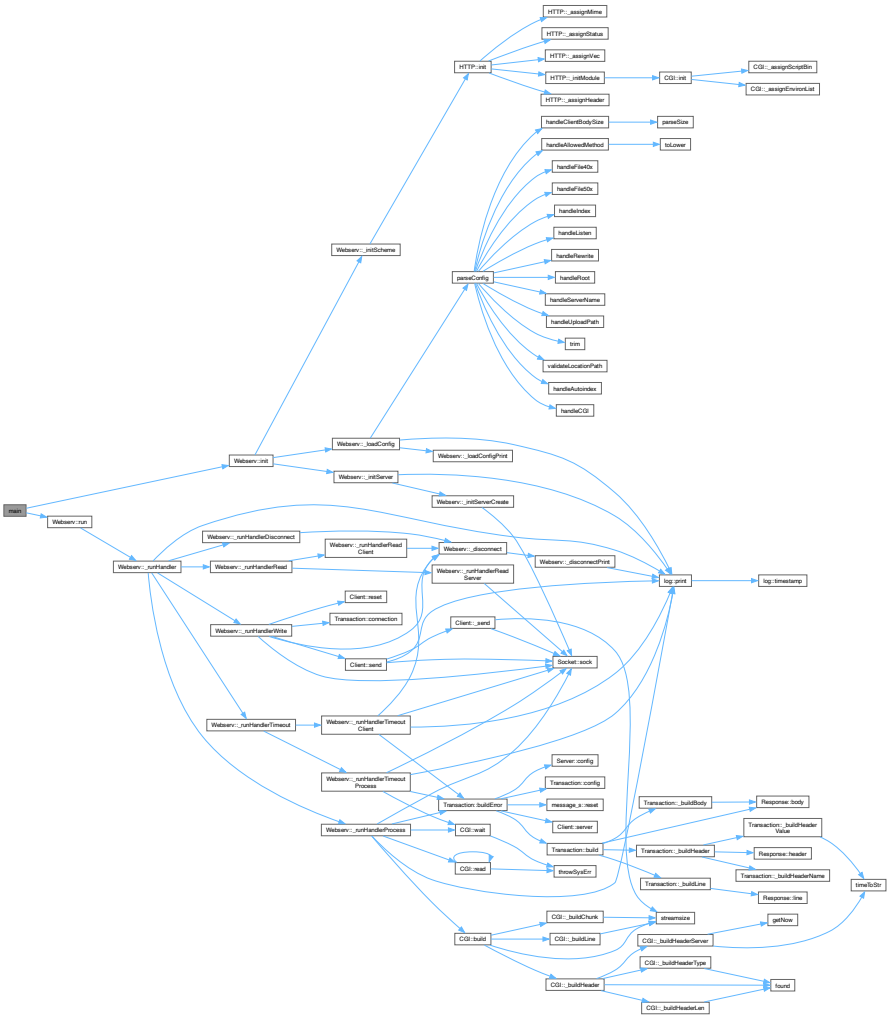
- int main (int argc, char ∗argv[ ])

7.51.1 Function Documentation

7.51.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

Here is the call graph for this function:



Index