

JWT Auth Server

Создано системой Doxygen 1.13.2

1 Алфавитный указатель пространств имен	1
1.1 Пространства имен	1
2 Иерархический список классов	2
2.1 Иерархия классов	2
3 Алфавитный указатель классов	4
3.1 Классы	4
4 Список файлов	6
4.1 Файлы	6
5 Пространства имен	7
5.1 Пространство имен anonymous_namespace{SHA256.cpp}	7
5.1.1 Функции	8
5.1.1.1 big_sigma0()	8
5.1.1.2 big_sigma1()	8
5.1.1.3 ch()	9
5.1.1.4 maj()	9
5.1.1.5 pad()	9
5.1.1.6 rotr()	10
5.1.1.7 small_sigma0()	10
5.1.1.8 small_sigma1()	11
5.1.1.9 to_uint32()	11
5.1.2 Переменные	12
5.1.2.1 k	12
5.2 Пространство имен httpplib	12
5.2.1 Типы	14
5.2.1.1 ContentProvider	14
5.2.1.2 ContentProviderResourceReleaser	14
5.2.1.3 ContentProviderWithoutLength	14
5.2.1.4 ContentReceiver	15
5.2.1.5 ContentReceiverWithProgress	15
5.2.1.6 Headers	15
5.2.1.7 Logger	15
5.2.1.8 Match	15
5.2.1.9 MultipartContentHeader	15
5.2.1.10 MultipartFormDataItems	16
5.2.1.11 MultipartFormDataMap	16
5.2.1.12 MultipartFormDataProviderItems	16
5.2.1.13 Params	16
5.2.1.14 Progress	16
5.2.1.15 Range	16
5.2.1.16 Ranges	16
5.2.1.17 ResponseHandler	16

5.2.1.18 SocketOptions	16
5.2.2 Перечисления	16
5.2.2.1 Error	16
5.2.2.2 SSLVerifierResponse	17
5.2.2.3 StatusCode	18
5.2.3 Функции	20
5.2.3.1 append_query_params()	20
5.2.3.2 default_socket_options()	21
5.2.3.3 get_bearer_token_auth()	21
5.2.3.4 hosted_at() [1/2]	22
5.2.3.5 hosted_at() [2/2]	23
5.2.3.6 make_basic_authentication_header()	23
5.2.3.7 make_bearer_token_authentication_header()	24
5.2.3.8 make_range_header()	25
5.2.3.9 operator<<()	26
5.2.3.10 status_message()	26
5.2.3.11 to_string()	27
5.3 Пространство имен <code>httpplib::detail</code>	28
5.3.1 Перечисления	32
5.3.1.1 EncodingType	32
5.3.2 Функции	32
5.3.2.1 base64_encode()	32
5.3.2.2 bind_ip_address()	33
5.3.2.3 calc_actual_timeout()	33
5.3.2.4 can_compress_content_type()	34
5.3.2.5 close_socket()	35
5.3.2.6 create_client_socket()	36
5.3.2.7 create_socket()	38
5.3.2.8 decode_url()	41
5.3.2.9 divide() [1/2]	42
5.3.2.10 divide() [2/2]	42
5.3.2.11 duration_to_sec_and_usec()	43
5.3.2.12 encode_query_param()	43
5.3.2.13 encode_url()	44
5.3.2.14 encoding_type()	45
5.3.2.15 escape_abstract_namespace_unix_domain()	46
5.3.2.16 expect_content()	46
5.3.2.17 file_extension()	47
5.3.2.18 find_content_type()	47
5.3.2.19 from_hex_to_i()	48
5.3.2.20 from_i_to_hex()	49
5.3.2.21 get_header_value()	50
5.3.2.22 get_header_value_u64() [1/2]	51

5.3.2.23 get_header_value_u64() [2/2]	51
5.3.2.24 get_ip_and_port()	52
5.3.2.25 get_local_ip_and_port()	53
5.3.2.26 get_multipart_ranges_data_length()	54
5.3.2.27 get_range_offset_and_length()	55
5.3.2.28 get_remote_ip_and_port()	56
5.3.2.29 handle_EINTR()	57
5.3.2.30 has_crlf()	57
5.3.2.31 has_header()	58
5.3.2.32 if2ip()	59
5.3.2.33 is_chunked_transfer_encoding()	60
5.3.2.34 is_connection_error()	61
5.3.2.35 is_hex()	62
5.3.2.36 is_multipart_boundary_chars_valid()	62
5.3.2.37 is_numeric()	63
5.3.2.38 is_socket_alive()	63
5.3.2.39 is_space_or_tab()	64
5.3.2.40 is_valid_path()	65
5.3.2.41 keep_alive()	66
5.3.2.42 make_content_range_header_field()	66
5.3.2.43 make_multipart_data_boundary()	67
5.3.2.44 make_multipart_ranges_data()	68
5.3.2.45 make_unique() [1/2]	69
5.3.2.46 make_unique() [2/2]	69
5.3.2.47 params_to_query_str()	70
5.3.2.48 parse_disposition_params()	70
5.3.2.49 parse_header()	71
5.3.2.50 parse_multipart_boundary()	72
5.3.2.51 parse_query_text() [1/2]	73
5.3.2.52 parse_query_text() [2/2]	74
5.3.2.53 parse_range_header()	75
5.3.2.54 parse_www_authenticate()	76
5.3.2.55 poll_wrapper()	77
5.3.2.56 prepare_content_receiver()	78
5.3.2.57 process_client_socket()	79
5.3.2.58 process_multipart_ranges_data()	80
5.3.2.59 process_server_socket()	81
5.3.2.60 process_server_socket_core()	82
5.3.2.61 random_string()	83
5.3.2.62 range_error()	84
5.3.2.63 read_content()	85
5.3.2.64 read_content_chunked()	86
5.3.2.65 read_content_with_length()	88

5.3.2.66 read_content_without_length()	89
5.3.2.67 read_headers()	90
5.3.2.68 read_socket()	91
5.3.2.69 redirect()	92
5.3.2.70 select_impl()	93
5.3.2.71 select_read()	94
5.3.2.72 select_write()	96
5.3.2.73 send_socket()	96
5.3.2.74 serialize_multipart_formdata()	97
5.3.2.75 serialize_multipart_formdata_finish()	98
5.3.2.76 serialize_multipart_formdata_get_content_type()	99
5.3.2.77 serialize_multipart_formdata_item_begin()	100
5.3.2.78 serialize_multipart_formdata_item_end()	101
5.3.2.79 set_nonblocking()	102
5.3.2.80 set_socket_opt()	103
5.3.2.81 set_socket_optImpl()	103
5.3.2.82 set_socket_opt_time()	104
5.3.2.83 shutdown_socket()	105
5.3.2.84 skip_content_with_length()	106
5.3.2.85 split() [1/2]	107
5.3.2.86 split() [2/2]	108
5.3.2.87 str2tag()	109
5.3.2.88 str2tag_core()	109
5.3.2.89 str_len()	110
5.3.2.90 to_utf8()	110
5.3.2.91 trim()	111
5.3.2.92 trim_copy()	112
5.3.2.93 trim_double_quotes_copy()	113
5.3.2.94 unescape_abstract_namespace_unix_domain()	113
5.3.2.95 wait_until_socket_is_ready()	114
5.3.2.96 write_content() [1/2]	115
5.3.2.97 write_content() [2/2]	116
5.3.2.98 write_content_chunked() [1/2]	117
5.3.2.99 write_content_chunked() [2/2]	117
5.3.2.100 write_content_without_length()	119
5.3.2.101 write_data()	120
5.3.2.102 write_headers()	121
5.3.2.103 write_multipart_ranges_data()	122
5.3.2.104 write_request_line()	123
5.3.2.105 write_response_line()	124
5.4 Пространство имен <code>httpplib::detail::case_ignore</code>	125
5.4.1 Функции	125
5.4.1.1 equal()	125

5.4.1.2 <code>to_lower()</code>	125
5.5 Пространство имен <code>httpplib::detail::fields</code>	126
5.5.1 Функции	126
5.5.1.1 <code>is_field_content()</code>	126
5.5.1.2 <code>is_field_name()</code>	128
5.5.1.3 <code>is_field_value()</code>	129
5.5.1.4 <code>is_field_vchar()</code>	130
5.5.1.5 <code>is_obs_text()</code>	131
5.5.1.6 <code>is_token()</code>	131
5.5.1.7 <code>is_token_char()</code>	132
5.5.1.8 <code>is_vchar()</code>	133
5.6 Пространство имен <code>httpplib::detail::udl</code>	134
5.6.1 Функции	134
5.6.1.1 <code>operator_t()</code>	134
6 Классы	136
6.1 Класс <code>Base64URL</code>	136
6.1.1 Подробное описание	137
6.1.2 Методы	137
6.1.2.1 <code>decode()</code>	137
6.1.2.2 <code>encode()</code>	138
6.2 Класс <code>BigInt</code>	140
6.2.1 Подробное описание	142
6.2.2 Конструктор(ы)	142
6.2.2.1 <code>BigInt() [1/4]</code>	142
6.2.2.2 <code>BigInt() [2/4]</code>	143
6.2.2.3 <code>BigInt() [3/4]</code>	143
6.2.2.4 <code>BigInt() [4/4]</code>	144
6.2.3 Методы	145
6.2.3.1 <code>compareAbs()</code>	145
6.2.3.2 <code>gcd()</code>	146
6.2.3.3 <code>isNegative()</code>	147
6.2.3.4 <code>isZero()</code>	147
6.2.3.5 <code>modPow()</code>	147
6.2.3.6 <code>operator"!="()</code>	149
6.2.3.7 <code>operator%()</code>	149
6.2.3.8 <code>operator*()</code>	150
6.2.3.9 <code>operator+()</code>	151
6.2.3.10 <code>operator-() [1/2]</code>	152
6.2.3.11 <code>operator-() [2/2]</code>	152
6.2.3.12 <code>operator/()</code>	153
6.2.3.13 <code>operator<()</code>	155
6.2.3.14 <code>operator<=()</code>	155

6.2.3.15 operator==()	156
6.2.3.16 operator>()	156
6.2.3.17 operator>=()	157
6.2.3.18 toString() [1/2]	157
6.2.3.19 toString() [2/2]	158
6.2.3.20 trim()	159
6.2.4 Данные класса	159
6.2.4.1 digits	159
6.2.4.2 negative	160
6.3 Класс <code>httpplib::detail::BufferStream</code>	160
6.3.1 Подробное описание	163
6.3.2 Конструктор(ы)	163
6.3.2.1 BufferStream()	163
6.3.2.2 ~BufferStream()	163
6.3.3 Методы	163
6.3.3.1 duration()	163
6.3.3.2 get_buffer()	164
6.3.3.3 get_local_ip_and_port()	164
6.3.3.4 get_remote_ip_and_port()	164
6.3.3.5 is_readable()	164
6.3.3.6 read()	165
6.3.3.7 socket()	165
6.3.3.8 wait_readable()	165
6.3.3.9 wait_writable()	165
6.3.3.10 write()	165
6.3.4 Данные класса	166
6.3.4.1 buffer	166
6.3.4.2 position	166
6.4 Класс <code>httpplib::Client</code>	166
6.4.1 Подробное описание	170
6.4.2 Конструктор(ы)	170
6.4.2.1 Client() [1/5]	170
6.4.2.2 Client() [2/5]	171
6.4.2.3 Client() [3/5]	172
6.4.2.4 Client() [4/5]	172
6.4.2.5 Client() [5/5]	173
6.4.2.6 ~Client()	173
6.4.3 Методы	173
6.4.3.1 Delete() [1/10]	173
6.4.3.2 Delete() [2/10]	174
6.4.3.3 Delete() [3/10]	174
6.4.3.4 Delete() [4/10]	174
6.4.3.5 Delete() [5/10]	174

6.4.3.6 Delete() [6/10]	175
6.4.3.7 Delete() [7/10]	175
6.4.3.8 Delete() [8/10]	175
6.4.3.9 Delete() [9/10]	175
6.4.3.10 Delete() [10/10]	176
6.4.3.11 Get() [1/15]	176
6.4.3.12 Get() [2/15]	176
6.4.3.13 Get() [3/15]	176
6.4.3.14 Get() [4/15]	176
6.4.3.15 Get() [5/15]	177
6.4.3.16 Get() [6/15]	177
6.4.3.17 Get() [7/15]	177
6.4.3.18 Get() [8/15]	177
6.4.3.19 Get() [9/15]	178
6.4.3.20 Get() [10/15]	178
6.4.3.21 Get() [11/15]	178
6.4.3.22 Get() [12/15]	178
6.4.3.23 Get() [13/15]	179
6.4.3.24 Get() [14/15]	179
6.4.3.25 Get() [15/15]	179
6.4.3.26 Head() [1/2]	179
6.4.3.27 Head() [2/2]	179
6.4.3.28 host()	180
6.4.3.29 is_socket_open()	180
6.4.3.30 is_valid()	180
6.4.3.31 operator=()	181
6.4.3.32 Options() [1/2]	181
6.4.3.33 Options() [2/2]	181
6.4.3.34 Patch() [1/13]	181
6.4.3.35 Patch() [2/13]	182
6.4.3.36 Patch() [3/13]	182
6.4.3.37 Patch() [4/13]	182
6.4.3.38 Patch() [5/13]	182
6.4.3.39 Patch() [6/13]	183
6.4.3.40 Patch() [7/13]	183
6.4.3.41 Patch() [8/13]	183
6.4.3.42 Patch() [9/13]	183
6.4.3.43 Patch() [10/13]	184
6.4.3.44 Patch() [11/13]	184
6.4.3.45 Patch() [12/13]	184
6.4.3.46 Patch() [13/13]	184
6.4.3.47 port()	185
6.4.3.48 Post() [1/20]	185

6.4.3.49 Post() [2/20]	185
6.4.3.50 Post() [3/20]	186
6.4.3.51 Post() [4/20]	186
6.4.3.52 Post() [5/20]	186
6.4.3.53 Post() [6/20]	186
6.4.3.54 Post() [7/20]	187
6.4.3.55 Post() [8/20]	187
6.4.3.56 Post() [9/20]	187
6.4.3.57 Post() [10/20]	187
6.4.3.58 Post() [11/20]	188
6.4.3.59 Post() [12/20]	188
6.4.3.60 Post() [13/20]	188
6.4.3.61 Post() [14/20]	188
6.4.3.62 Post() [15/20]	189
6.4.3.63 Post() [16/20]	189
6.4.3.64 Post() [17/20]	189
6.4.3.65 Post() [18/20]	189
6.4.3.66 Post() [19/20]	189
6.4.3.67 Post() [20/20]	190
6.4.3.68 Put() [1/19]	190
6.4.3.69 Put() [2/19]	190
6.4.3.70 Put() [3/19]	190
6.4.3.71 Put() [4/19]	191
6.4.3.72 Put() [5/19]	191
6.4.3.73 Put() [6/19]	191
6.4.3.74 Put() [7/19]	191
6.4.3.75 Put() [8/19]	192
6.4.3.76 Put() [9/19]	192
6.4.3.77 Put() [10/19]	192
6.4.3.78 Put() [11/19]	192
6.4.3.79 Put() [12/19]	193
6.4.3.80 Put() [13/19]	193
6.4.3.81 Put() [14/19]	193
6.4.3.82 Put() [15/19]	193
6.4.3.83 Put() [16/19]	193
6.4.3.84 Put() [17/19]	194
6.4.3.85 Put() [18/19]	194
6.4.3.86 Put() [19/19]	194
6.4.3.87 send() [1/2]	194
6.4.3.88 send() [2/2]	194
6.4.3.89 set_address_family()	195
6.4.3.90 set_basic_auth()	195
6.4.3.91 set_bearer_token_auth()	195

6.4.3.92 set_compress()	195
6.4.3.93 set_connection_timeout() [1/2]	195
6.4.3.94 set_connection_timeout() [2/2]	196
6.4.3.95 set_decompress()	196
6.4.3.96 set_default_headers()	196
6.4.3.97 set_follow_location()	196
6.4.3.98 set_header_writer()	196
6.4.3.99 set_hostname_addr_map()	197
6.4.3.100 set_interface()	197
6.4.3.101 set_keep_alive()	197
6.4.3.102 set_logger()	197
6.4.3.103 set_max_timeout() [1/2]	197
6.4.3.104 set_max_timeout() [2/2]	198
6.4.3.105 set_proxy()	198
6.4.3.106 set_proxy_basic_auth()	198
6.4.3.107 set_proxy_bearer_token_auth()	199
6.4.3.108 set_read_timeout() [1/2]	199
6.4.3.109 set_read_timeout() [2/2]	199
6.4.3.110 set_socket_options()	199
6.4.3.111 set_tcp_nodelay()	199
6.4.3.112 set_url_encode()	200
6.4.3.113 set_write_timeout() [1/2]	200
6.4.3.114 set_write_timeout() [2/2]	200
6.4.3.115 socket()	200
6.4.3.116 stop()	200
6.4.4 Данные класса	200
6.4.4.1 cli_	200
6.5 Класс http://ClientImpl	201
6.5.1 Подробное описание	206
6.5.2 Конструктор(ы)	207
6.5.2.1 ClientImpl() [1/3]	207
6.5.2.2 ClientImpl() [2/3]	207
6.5.2.3 ClientImpl() [3/3]	208
6.5.2.4 ~ClientImpl()	209
6.5.3 Методы	209
6.5.3.1 adjust_host_string()	209
6.5.3.2 close_socket()	210
6.5.3.3 copy_settings()	211
6.5.3.4 create_and_connect_socket()	212
6.5.3.5 create_client_socket()	214
6.5.3.6 Delete() [1/10]	216
6.5.3.7 Delete() [2/10]	217
6.5.3.8 Delete() [3/10]	218

6.5.3.9 Delete() [4/10]	218
6.5.3.10 Delete() [5/10]	219
6.5.3.11 Delete() [6/10]	219
6.5.3.12 Delete() [7/10]	220
6.5.3.13 Delete() [8/10]	221
6.5.3.14 Delete() [9/10]	221
6.5.3.15 Delete() [10/10]	222
6.5.3.16 Get() [1/15]	222
6.5.3.17 Get() [2/15]	223
6.5.3.18 Get() [3/15]	224
6.5.3.19 Get() [4/15]	224
6.5.3.20 Get() [5/15]	225
6.5.3.21 Get() [6/15]	225
6.5.3.22 Get() [7/15]	226
6.5.3.23 Get() [8/15]	227
6.5.3.24 Get() [9/15]	227
6.5.3.25 Get() [10/15]	228
6.5.3.26 Get() [11/15]	228
6.5.3.27 Get() [12/15]	229
6.5.3.28 Get() [13/15]	229
6.5.3.29 Get() [14/15]	230
6.5.3.30 Get() [15/15]	230
6.5.3.31 get_multipart_content_provider()	231
6.5.3.32 handle_request()	232
6.5.3.33 Head() [1/2]	234
6.5.3.34 Head() [2/2]	235
6.5.3.35 host()	236
6.5.3.36 is_socket_open()	236
6.5.3.37 is_ssl()	237
6.5.3.38 is_valid()	237
6.5.3.39 Options() [1/2]	238
6.5.3.40 Options() [2/2]	238
6.5.3.41 Patch() [1/13]	239
6.5.3.42 Patch() [2/13]	240
6.5.3.43 Patch() [3/13]	241
6.5.3.44 Patch() [4/13]	241
6.5.3.45 Patch() [5/13]	242
6.5.3.46 Patch() [6/13]	243
6.5.3.47 Patch() [7/13]	243
6.5.3.48 Patch() [8/13]	244
6.5.3.49 Patch() [9/13]	244
6.5.3.50 Patch() [10/13]	245
6.5.3.51 Patch() [11/13]	245

6.5.3.52 Patch() [12/13]	246
6.5.3.53 Patch() [13/13]	246
6.5.3.54 port()	247
6.5.3.55 Post() [1/20]	247
6.5.3.56 Post() [2/20]	248
6.5.3.57 Post() [3/20]	249
6.5.3.58 Post() [4/20]	249
6.5.3.59 Post() [5/20]	250
6.5.3.60 Post() [6/20]	251
6.5.3.61 Post() [7/20]	251
6.5.3.62 Post() [8/20]	252
6.5.3.63 Post() [9/20]	253
6.5.3.64 Post() [10/20]	253
6.5.3.65 Post() [11/20]	254
6.5.3.66 Post() [12/20]	254
6.5.3.67 Post() [13/20]	255
6.5.3.68 Post() [14/20]	255
6.5.3.69 Post() [15/20]	256
6.5.3.70 Post() [16/20]	256
6.5.3.71 Post() [17/20]	257
6.5.3.72 Post() [18/20]	257
6.5.3.73 Post() [19/20]	258
6.5.3.74 Post() [20/20]	258
6.5.3.75 process_request()	259
6.5.3.76 process_socket()	261
6.5.3.77 Put() [1/19]	262
6.5.3.78 Put() [2/19]	264
6.5.3.79 Put() [3/19]	264
6.5.3.80 Put() [4/19]	265
6.5.3.81 Put() [5/19]	265
6.5.3.82 Put() [6/19]	266
6.5.3.83 Put() [7/19]	267
6.5.3.84 Put() [8/19]	267
6.5.3.85 Put() [9/19]	268
6.5.3.86 Put() [10/19]	268
6.5.3.87 Put() [11/19]	269
6.5.3.88 Put() [12/19]	270
6.5.3.89 Put() [13/19]	270
6.5.3.90 Put() [14/19]	271
6.5.3.91 Put() [15/19]	271
6.5.3.92 Put() [16/19]	272
6.5.3.93 Put() [17/19]	272
6.5.3.94 Put() [18/19]	273

6.5.3.95 Put() [19/19]	273
6.5.3.96 read_response_line()	274
6.5.3.97 redirect()	275
6.5.3.98 send() [1/2]	276
6.5.3.99 send() [2/2]	277
6.5.3.100 send_() [1/2]	278
6.5.3.101 send_() [2/2]	279
6.5.3.102 send_with_content_provider() [1/2]	281
6.5.3.103 send_with_content_provider() [2/2]	282
6.5.3.104 set_address_family()	284
6.5.3.105 set_basic_auth()	285
6.5.3.106 set_bearer_token_auth()	285
6.5.3.107 set_compress()	285
6.5.3.108 set_connection_timeout() [1/2]	285
6.5.3.109 set_connection_timeout() [2/2]	286
6.5.3.110 set_decompress()	286
6.5.3.111 set_default_headers()	286
6.5.3.112 set_follow_location()	286
6.5.3.113 set_header_writer()	287
6.5.3.114 set_hostname_addr_map()	287
6.5.3.115 set_interface()	287
6.5.3.116 set_ipv6_v6only()	287
6.5.3.117 set_keep_alive()	287
6.5.3.118 set_logger()	287
6.5.3.119 set_max_timeout() [1/2]	288
6.5.3.120 set_max_timeout() [2/2]	288
6.5.3.121 set_proxy()	289
6.5.3.122 set_proxy_basic_auth()	289
6.5.3.123 set_proxy_bearer_token_auth()	289
6.5.3.124 set_read_timeout() [1/2]	290
6.5.3.125 set_read_timeout() [2/2]	290
6.5.3.126 set_socket_options()	291
6.5.3.127 set_tcp_nodelay()	291
6.5.3.128 set_url_encode()	291
6.5.3.129 set_write_timeout() [1/2]	291
6.5.3.130 set_write_timeout() [2/2]	292
6.5.3.131 shutdown_socket()	292
6.5.3.132 shutdown_ssl()	293
6.5.3.133 socket()	294
6.5.3.134 stop()	295
6.5.3.135 write_content_with_provider()	296
6.5.3.136 write_request()	297
6.5.4 Данные класса	300

6.5.4.1 <code>addr_map_</code>	300
6.5.4.2 <code>address_family_</code>	300
6.5.4.3 <code>basic_auth_password_</code>	300
6.5.4.4 <code>basic_auth_username_</code>	300
6.5.4.5 <code>bearer_token_auth_token_</code>	300
6.5.4.6 <code>client_cert_path_</code>	300
6.5.4.7 <code>client_key_path_</code>	301
6.5.4.8 <code>compress_</code>	301
6.5.4.9 <code>connection_timeout_sec_</code>	301
6.5.4.10 <code>connection_timeout_usec_</code>	301
6.5.4.11 <code>decompress_</code>	301
6.5.4.12 <code>default_headers_</code>	301
6.5.4.13 <code>follow_location_</code>	301
6.5.4.14 <code>header_writer_</code>	302
6.5.4.15 <code>host_</code>	302
6.5.4.16 <code>host_and_port_</code>	302
6.5.4.17 <code>interface_</code>	302
6.5.4.18 <code>ipv6_v6only_</code>	302
6.5.4.19 <code>keep_alive_</code>	302
6.5.4.20 <code>logger_</code>	302
6.5.4.21 <code>max_timeout_msec_</code>	303
6.5.4.22 <code>port_</code>	303
6.5.4.23 <code>proxy_basic_auth_password_</code>	303
6.5.4.24 <code>proxy_basic_auth_username_</code>	303
6.5.4.25 <code>proxy_bearer_token_auth_token_</code>	303
6.5.4.26 <code>proxy_host_</code>	303
6.5.4.27 <code>proxy_port_</code>	303
6.5.4.28 <code>read_timeout_sec_</code>	303
6.5.4.29 <code>read_timeout_usec_</code>	304
6.5.4.30 <code>request_mutex_</code>	304
6.5.4.31 <code>socket_</code>	304
6.5.4.32 <code>socket_mutex_</code>	304
6.5.4.33 <code>socket_options_</code>	304
6.5.4.34 <code>socket_requests_are_from_thread_</code>	304
6.5.4.35 <code>socket_requests_in_flight_</code>	304
6.5.4.36 <code>socket_should_be_closed_when_request_is_done_</code>	304
6.5.4.37 <code>tcp_nodelay_</code>	305
6.5.4.38 <code>url_encode_</code>	305
6.5.4.39 <code>write_timeout_sec_</code>	305
6.5.4.40 <code>write_timeout_usec_</code>	305
6.6 Класс <code>httpplib::detail::compressor</code>	305
6.6.1 Подробное описание	306
6.6.2 Определения типов	306

6.6.2.1 Callback	306
6.6.3 Конструктор(ы)	306
6.6.3.1 ~compressor()	306
6.6.4 Методы	307
6.6.4.1 compress()	307
6.7 Класс <code>httpplib::detail::ContentProviderAdapter</code>	307
6.7.1 Подробное описание	308
6.7.2 Конструктор(ы)	308
6.7.2.1 ContentProviderAdapter()	308
6.7.3 Методы	308
6.7.3.1 operator()()	308
6.7.4 Данные класса	308
6.7.4.1 content_provider_	308
6.8 Класс <code>httpplib::ContentReader</code>	309
6.8.1 Подробное описание	309
6.8.2 Определения типов	309
6.8.2.1 MultipartReader	309
6.8.2.2 Reader	310
6.8.3 Конструктор(ы)	310
6.8.3.1 ContentReader()	310
6.8.4 Методы	310
6.8.4.1 operator()() [1/2]	310
6.8.4.2 operator()() [2/2]	310
6.8.5 Данные класса	310
6.8.5.1 multipart_reader_	310
6.8.5.2 reader_	311
6.9 Класс <code>httpplib::DataSink::data_sink_streambuf</code>	311
6.9.1 Подробное описание	313
6.9.2 Конструктор(ы)	313
6.9.2.1 data_sink_streambuf()	313
6.9.3 Методы	313
6.9.3.1 xsputn()	313
6.9.4 Данные класса	313
6.9.4.1 sink_	313
6.10 Класс <code>Database</code>	314
6.10.1 Подробное описание	314
6.10.2 Методы	315
6.10.2.1 addUser()	315
6.10.2.2 blacklistToken()	316
6.10.2.3 cleanupBlacklist()	316
6.10.2.4 getUser()	317
6.10.2.5 init()	318
6.10.2.6 isTokenBlacklisted()	319

6.11 Класс <code>httpplib::DataSink</code>	320
6.11.1 Подробное описание	321
6.11.2 Конструктор(ы)	321
6.11.2.1 <code>DataSink()</code> [1/3]	321
6.11.2.2 <code>DataSink()</code> [2/3]	322
6.11.2.3 <code>DataSink()</code> [3/3]	322
6.11.3 Методы	322
6.11.3.1 <code>operator=()</code> [1/2]	322
6.11.3.2 <code>operator=()</code> [2/2]	323
6.11.4 Данные класса	323
6.11.4.1 <code>done</code>	323
6.11.4.2 <code>done_with_trailer</code>	323
6.11.4.3 <code>is_writable</code>	323
6.11.4.4 <code>os</code>	323
6.11.4.5 <code>sb_</code>	323
6.11.4.6 <code>write</code>	324
6.12 Класс <code>httpplib::detail::decompressor</code>	324
6.12.1 Подробное описание	324
6.12.2 Определения типов	325
6.12.2.1 <code>Callback</code>	325
6.12.3 Конструктор(ы)	325
6.12.3.1 <code>~decompressor()</code>	325
6.12.4 Методы	325
6.12.4.1 <code>decompress()</code>	325
6.12.4.2 <code>is_valid()</code>	325
6.13 Структура <code>httpplib::detail::case_ignore::equal_to</code>	326
6.13.1 Подробное описание	326
6.13.2 Методы	326
6.13.2.1 <code>operator()()</code>	326
6.14 Структура <code>httpplib::detail::FileStat</code>	327
6.14.1 Подробное описание	327
6.14.2 Конструктор(ы)	327
6.14.2.1 <code>FileStat()</code>	327
6.14.3 Методы	328
6.14.3.1 <code>is_dir()</code>	328
6.14.3.2 <code>is_file()</code>	328
6.14.4 Данные класса	328
6.14.4.1 <code>ret_</code>	328
6.14.4.2 <code>st_</code>	328
6.15 Структура <code>httpplib::detail::case_ignore::hash</code>	329
6.15.1 Подробное описание	329
6.15.2 Методы	329
6.15.2.1 <code>hash_core()</code>	329

6.15.2.2 operator()()	330
6.16 Класс HttpServer	331
6.16.1 Подробное описание	331
6.16.2 Методы	331
6.16.2.1 start()	331
6.17 Класс JWT	336
6.17.1 Подробное описание	337
6.17.2 Методы	338
6.17.2.1 createAccessToken()	338
6.17.2.2 createRefreshToken()	339
6.17.2.3 verifyAccessToken()	341
6.17.2.4 verifyRefreshToken()	342
6.18 Класс KeyStorage	344
6.18.1 Подробное описание	345
6.18.2 Методы	345
6.18.2.1 loadKeys()	345
6.18.2.2 saveKeys()	347
6.19 Класс httplib::detail::MatcherBase	348
6.19.1 Подробное описание	349
6.19.2 Конструктор(ы)	349
6.19.2.1 ~MatcherBase()	349
6.19.3 Методы	349
6.19.3.1 match()	349
6.20 Класс httplib::detail::mmap	349
6.20.1 Подробное описание	350
6.20.2 Конструктор(ы)	350
6.20.2.1 mmap()	350
6.20.2.2 ~mmap()	351
6.20.3 Методы	351
6.20.3.1 close()	351
6.20.3.2 data()	352
6.20.3.3 is_open()	352
6.20.3.4 open()	353
6.20.3.5 size()	354
6.20.4 Данные класса	355
6.20.4.1 addr_	355
6.20.4.2 fd_	355
6.20.4.3 is_open_empty_file	355
6.20.4.4 size_	355
6.21 Структура httplib::Server::MountPointEntry	355
6.21.1 Подробное описание	356
6.21.2 Данные класса	356
6.21.2.1 base_dir	356

6.21.2.2 headers	356
6.21.2.3 mount_point	356
6.22 Структура <code>httpplib::MultipartFormData</code>	356
6.22.1 Подробное описание	357
6.22.2 Данные класса	357
6.22.2.1 content	357
6.22.2.2 content_type	357
6.22.2.3 filename	357
6.22.2.4 name	357
6.23 Класс <code>httpplib::detail::MultipartFormDataParser</code>	358
6.23.1 Подробное описание	359
6.23.2 Конструктор(ы)	359
6.23.2.1 <code>MultipartFormDataParser()</code>	359
6.23.3 Методы	360
6.23.3.1 <code>buf_append()</code>	360
6.23.3.2 <code>buf_data()</code>	360
6.23.3.3 <code>buf_erase()</code>	361
6.23.3.4 <code>buf_find()</code>	361
6.23.3.5 <code>buf_head()</code>	362
6.23.3.6 <code>buf_size()</code>	362
6.23.3.7 <code>buf_start_with()</code>	362
6.23.3.8 <code>clear_file_info()</code>	363
6.23.3.9 <code>is_valid()</code>	363
6.23.3.10 <code>parse()</code>	364
6.23.3.11 <code>set_boundary()</code>	366
6.23.3.12 <code>start_with()</code>	367
6.23.3.13 <code>start_with_case_ignore()</code>	367
6.23.4 Данные класса	368
6.23.4.1 <code>boundary_</code>	368
6.23.4.2 <code>buf_</code>	368
6.23.4.3 <code>buf_epos_</code>	368
6.23.4.4 <code>buf_spos_</code>	368
6.23.4.5 <code>crlf_</code>	368
6.23.4.6 <code>crlf_dash_boundary_</code>	368
6.23.4.7 <code>dash_</code>	368
6.23.4.8 <code>dash_boundary_crlf_</code>	369
6.23.4.9 <code>file_</code>	369
6.23.4.10 <code>is_valid_</code>	369
6.23.4.11 <code>state_</code>	369
6.24 Структура <code>httpplib::MultipartFormDataProvider</code>	369
6.24.1 Подробное описание	370
6.24.2 Данные класса	370
6.24.2.1 <code>content_type</code>	370

6.24.2.2 filename	370
6.24.2.3 name	370
6.24.2.4 provider	370
6.25 Класс <code>httpplib::detail::nocompressor</code>	371
6.25.1 Подробное описание	372
6.25.2 Конструктор(ы)	372
6.25.2.1 <code>~nocompressor()</code>	372
6.25.3 Методы	372
6.25.3.1 <code>compress()</code>	372
6.26 Класс <code>PasswordEncryptor</code>	373
6.26.1 Подробное описание	373
6.26.2 Методы	373
6.26.2.1 <code>hashPassword()</code>	373
6.27 Класс <code>httpplib::detail::PathParamMatcher</code>	374
6.27.1 Подробное описание	376
6.27.2 Конструктор(ы)	376
6.27.2.1 <code>PathParamMatcher()</code>	376
6.27.3 Методы	377
6.27.3.1 <code>match()</code>	377
6.27.4 Данные класса	378
6.27.4.1 <code>param_names_</code>	378
6.27.4.2 <code>separator</code>	378
6.27.4.3 <code>static_fragments_</code>	378
6.28 Класс <code>httpplib::detail::RegexMatcher</code>	379
6.28.1 Подробное описание	380
6.28.2 Конструктор(ы)	380
6.28.2.1 <code>RegexMatcher()</code>	380
6.28.3 Методы	380
6.28.3.1 <code>match()</code>	380
6.28.4 Данные класса	380
6.28.4.1 <code>regex_</code>	380
6.29 Структура <code>httpplib::Request</code>	381
6.29.1 Подробное описание	382
6.29.2 Методы	382
6.29.2.1 <code>get_file_value()</code>	382
6.29.2.2 <code>get_file_values()</code>	383
6.29.2.3 <code>get_header_value()</code>	384
6.29.2.4 <code>get_header_value_count()</code>	385
6.29.2.5 <code>get_header_value_u64()</code>	385
6.29.2.6 <code>get_param_value()</code>	386
6.29.2.7 <code>get_param_value_count()</code>	387
6.29.2.8 <code>has_file()</code>	388
6.29.2.9 <code>has_header()</code>	388

6.29.2.10 has_param()	389
6.29.2.11 is_multipart_form_data()	390
6.29.2.12 set_header()	390
6.29.3 Даные класса	391
6.29.3.1 authorization_count_	391
6.29.3.2 body	392
6.29.3.3 content_length_	392
6.29.3.4 content_provider_	392
6.29.3.5 content_receiver	392
6.29.3.6 files	392
6.29.3.7 headers	392
6.29.3.8 is_chunked_content_provider_	392
6.29.3.9 is_connection_closed	392
6.29.3.10 local_addr	393
6.29.3.11 local_port	393
6.29.3.12 matches	393
6.29.3.13 method	393
6.29.3.14 params	393
6.29.3.15 path	393
6.29.3.16 path_params	393
6.29.3.17 progress	393
6.29.3.18 ranges	394
6.29.3.19 redirect_count_	394
6.29.3.20 remote_addr	394
6.29.3.21 remote_port	394
6.29.3.22 response_handler	394
6.29.3.23 start_time_	394
6.29.3.24 target	394
6.29.3.25 version	395
6.30 Структура http://Response	395
6.30.1 Подробное описание	397
6.30.2 Конструктор(ы)	397
6.30.2.1 Response() [1/3]	397
6.30.2.2 Response() [2/3]	398
6.30.2.3 Response() [3/3]	398
6.30.2.4 ~Response()	399
6.30.3 Методы	399
6.30.3.1 get_header_value()	399
6.30.3.2 get_header_value_count()	400
6.30.3.3 get_header_value_u64()	401
6.30.3.4 has_header()	401
6.30.3.5 operator=() [1/2]	402
6.30.3.6 operator=() [2/2]	402

6.30.3.7 set_chunked_content_provider()	403
6.30.3.8 set_content() [1/3]	403
6.30.3.9 set_content() [2/3]	404
6.30.3.10 set_content() [3/3]	405
6.30.3.11 set_content_provider() [1/2]	405
6.30.3.12 set_content_provider() [2/2]	406
6.30.3.13 set_file_content() [1/2]	406
6.30.3.14 set_file_content() [2/2]	407
6.30.3.15 set_header()	407
6.30.3.16 set_redirect()	408
6.30.4 Данные класса	409
6.30.4.1 body	409
6.30.4.2 content_length_	409
6.30.4.3 content_provider_	409
6.30.4.4 content_provider_resource_releaser_	409
6.30.4.5 content_provider_success_	409
6.30.4.6 file_content_content_type_	410
6.30.4.7 file_content_path_	410
6.30.4.8 headers	410
6.30.4.9 is_chunked_content_provider_	410
6.30.4.10 location	410
6.30.4.11 reason	410
6.30.4.12 status	410
6.30.4.13 version	411
6.31 Класс httpplib::Result	411
6.31.1 Подробное описание	412
6.31.2 Конструктор(ы)	412
6.31.2.1 Result() [1/2]	412
6.31.2.2 Result() [2/2]	412
6.31.3 Методы	412
6.31.3.1 error()	412
6.31.3.2 get_request_header_value()	413
6.31.3.3 get_request_header_value_count()	413
6.31.3.4 get_request_header_value_u64()	414
6.31.3.5 has_request_header()	415
6.31.3.6 operator bool()	415
6.31.3.7 operator"!=()	415
6.31.3.8 operator*() [1/2]	416
6.31.3.9 operator*() [2/2]	416
6.31.3.10 operator->() [1/2]	416
6.31.3.11 operator->() [2/2]	416
6.31.3.12 operator==()	416
6.31.3.13 value() [1/2]	416

6.31.3.14 value() [2/2]	416
6.31.4 Данные класса	417
6.31.4.1 err_	417
6.31.4.2 request_headers_	417
6.31.4.3 res_	417
6.32 Класс RSA	417
6.32.1 Подробное описание	418
6.32.2 Методы	418
6.32.2.1 decrypt()	418
6.32.2.2 encrypt()	419
6.32.2.3 generate_keys()	420
6.32.2.4 sign()	421
6.32.2.5 verify()	422
6.33 Структура RSAPrivateKey	424
6.33.1 Подробное описание	425
6.33.2 Данные класса	425
6.33.2.1 d	425
6.33.2.2 n	425
6.34 Структура RSAPublicKey	426
6.34.1 Подробное описание	427
6.34.2 Данные класса	427
6.34.2.1 e	427
6.34.2.2 n	427
6.35 Структура httpplib::detail::scope_exit	427
6.35.1 Подробное описание	428
6.35.2 Конструктор(ы)	428
6.35.2.1 scope_exit() [1/3]	428
6.35.2.2 scope_exit() [2/3]	429
6.35.2.3 ~scope_exit()	429
6.35.2.4 scope_exit() [3/3]	429
6.35.3 Методы	430
6.35.3.1 operator=() [1/2]	430
6.35.3.2 operator=() [2/2]	430
6.35.3.3 release()	430
6.35.4 Данные класса	430
6.35.4.1 execute_on_destruction	430
6.35.4.2 exit_function	431
6.36 Класс httpplib::Server	431
6.36.1 Подробное описание	436
6.36.2 Определения типов	436
6.36.2.1 ExceptionHandler	436
6.36.2.2 Expect100ContinueHandler	436
6.36.2.3 Handler	436

6.36.2.4 Handlers	436
6.36.2.5 HandlersForContentReader	436
6.36.2.6 HandlerWithContentReader	437
6.36.2.7 HandlerWithResponse	437
6.36.3 Перечисления	437
6.36.3.1 HandlerResponse	437
6.36.4 Конструктор(ы)	437
6.36.4.1 Server()	437
6.36.4.2 ~Server()	438
6.36.5 Методы	439
6.36.5.1 apply_ranges()	439
6.36.5.2 bind_internal()	441
6.36.5.3 bind_to_any_port()	442
6.36.5.4 bind_to_port()	443
6.36.5.5 create_server_socket()	443
6.36.5.6 decommission()	444
6.36.5.7 Delete() [1/2]	444
6.36.5.8 Delete() [2/2]	445
6.36.5.9 dispatch_request()	445
6.36.5.10 dispatch_request_for_content_reader()	446
6.36.5.11 Get()	446
6.36.5.12 handle_file_request()	447
6.36.5.13 is_running()	448
6.36.5.14 is_valid()	448
6.36.5.15 listen()	449
6.36.5.16 listen_after_bind()	449
6.36.5.17 listen_internal()	450
6.36.5.18 make_matcher()	451
6.36.5.19 Options()	452
6.36.5.20 parse_request_line()	453
6.36.5.21 Patch() [1/2]	454
6.36.5.22 Patch() [2/2]	455
6.36.5.23 Post() [1/2]	455
6.36.5.24 Post() [2/2]	456
6.36.5.25 process_and_close_socket()	456
6.36.5.26 process_request()	457
6.36.5.27 Put() [1/2]	461
6.36.5.28 Put() [2/2]	461
6.36.5.29 read_content()	462
6.36.5.30 read_content_core()	463
6.36.5.31 read_content_with_content_receiver()	465
6.36.5.32 remove_mount_point()	466
6.36.5.33 routing()	467

6.36.5.34 set_address_family()	468
6.36.5.35 set_base_dir()	469
6.36.5.36 set_default_file_mimetype()	469
6.36.5.37 set_default_headers()	470
6.36.5.38 set_error_handler()	470
6.36.5.39 set_error_handler_core() [1/2]	471
6.36.5.40 set_error_handler_core() [2/2]	471
6.36.5.41 set_exception_handler()	472
6.36.5.42 set_expect_100_continue_handler()	473
6.36.5.43 set_file_extension_and_mimetype_mapping()	473
6.36.5.44 set_file_request_handler()	473
6.36.5.45 set_header_writer()	474
6.36.5.46 set_idle_interval() [1/2]	474
6.36.5.47 set_idle_interval() [2/2]	475
6.36.5.48 set_ipv6_v6only()	475
6.36.5.49 set_keep_alive_max_count()	476
6.36.5.50 set_keep_alive_timeout()	476
6.36.5.51 set_logger()	477
6.36.5.52 set_mount_point()	477
6.36.5.53 set_payload_max_length()	478
6.36.5.54 set_post_routing_handler()	479
6.36.5.55 set_pre_routing_handler()	479
6.36.5.56 set_read_timeout() [1/2]	480
6.36.5.57 set_read_timeout() [2/2]	480
6.36.5.58 set_socket_options()	481
6.36.5.59 set_tcp_nodelay()	481
6.36.5.60 set_write_timeout() [1/2]	482
6.36.5.61 set_write_timeout() [2/2]	482
6.36.5.62 stop()	483
6.36.5.63 wait_until_ready()	483
6.36.5.64 write_content_with_provider()	484
6.36.5.65 write_response()	485
6.36.5.66 write_response_core()	486
6.36.5.67 write_response_with_content()	488
6.36.6 Данные класса	489
6.36.6.1 address_family_	489
6.36.6.2 base_dirs_	489
6.36.6.3 default_file_mimetype_	490
6.36.6.4 default_headers_	490
6.36.6.5 delete_handlers_	490
6.36.6.6 delete_handlers_for_content_reader_	490
6.36.6.7 error_handler_	490
6.36.6.8 exception_handler_	490

6.36.6.9 expect_100_continue_handler_	490
6.36.6.10 file_extension_and_mimetype_map_	490
6.36.6.11 file_request_handler_	491
6.36.6.12 get_handlers_	491
6.36.6.13 header_writer_	491
6.36.6.14 idle_interval_sec_	491
6.36.6.15 idle_interval_usec_	491
6.36.6.16 ipv6_v6only_	491
6.36.6.17 is_decommissioned	491
6.36.6.18 is_running_	492
6.36.6.19 keep_alive_max_count_	492
6.36.6.20 keep_alive_timeout_sec_	492
6.36.6.21 logger_	492
6.36.6.22 new_task_queue	492
6.36.6.23 options_handlers_	492
6.36.6.24 patch_handlers_	492
6.36.6.25 patch_handlers_for_content_reader_	492
6.36.6.26 payload_max_length_	493
6.36.6.27 post_handlers_	493
6.36.6.28 post_handlers_for_content_reader_	493
6.36.6.29 post_routing_handler_	493
6.36.6.30 pre_routing_handler_	493
6.36.6.31 put_handlers_	493
6.36.6.32 put_handlers_for_content_reader_	493
6.36.6.33 read_timeout_sec_	493
6.36.6.34 read_timeout_usec_	494
6.36.6.35 socket_options_	494
6.36.6.36 svr_sock_	494
6.36.6.37 tcp_nodelay_	494
6.36.6.38 write_timeout_sec_	494
6.36.6.39 write_timeout_usec_	494
6.37 Класс SHA256	495
6.37.1 Подробное описание	495
6.37.2 Методы	495
6.37.2.1 hash()	495
6.38 Структура httpplib::ClientImpl::Socket	497
6.38.1 Подробное описание	498
6.38.2 Методы	498
6.38.2.1 is_open()	498
6.38.3 Данные класса	498
6.38.3.1 sock	498
6.39 Класс httpplib::detail::SocketStream	498
6.39.1 Подробное описание	501

6.39.2 Конструктор(ы)	502
6.39.2.1 SocketStream()	502
6.39.2.2 ~SocketStream()	502
6.39.3 Методы	503
6.39.3.1 duration()	503
6.39.3.2 get_local_ip_and_port()	503
6.39.3.3 get_remote_ip_and_port()	504
6.39.3.4 is_readable()	504
6.39.3.5 read()	504
6.39.3.6 socket()	505
6.39.3.7 wait_readable()	505
6.39.3.8 wait_writable()	506
6.39.3.9 write()	507
6.39.4 Данние класса	507
6.39.4.1 max_timeout_msec_	507
6.39.4.2 read_buff_	507
6.39.4.3 read_buff_content_size_	508
6.39.4.4 read_buff_off_	508
6.39.4.5 read_buff_size_	508
6.39.4.6 read_timeout_sec_	508
6.39.4.7 read_timeout_usec_	508
6.39.4.8 sock_	508
6.39.4.9 start_time_	508
6.39.4.10 write_timeout_sec_	508
6.39.4.11 write_timeout_usec_	509
6.40 Класс httpplib::Stream	509
6.40.1 Подробное описание	511
6.40.2 Конструктор(ы)	511
6.40.2.1 ~Stream()	511
6.40.3 Методы	512
6.40.3.1 duration()	512
6.40.3.2 get_local_ip_and_port()	512
6.40.3.3 get_remote_ip_and_port()	512
6.40.3.4 is_readable()	512
6.40.3.5 read()	512
6.40.3.6 socket()	513
6.40.3.7 wait_readable()	513
6.40.3.8 wait_writable()	513
6.40.3.9 write() [1/3]	514
6.40.3.10 write() [2/3]	514
6.40.3.11 write() [3/3]	515
6.41 Класс httpplib::detail::stream_line_reader	515
6.41.1 Подробное описание	517

6.41.2 Конструктор(ы)	517
6.41.2.1 stream_line_reader()	517
6.41.3 Методы	517
6.41.3.1 append()	517
6.41.3.2 end_with_crlf()	518
6.41.3.3 getline()	518
6.41.3.4 ptr()	519
6.41.3.5 size()	520
6.41.4 Данные класса	520
6.41.4.1 fixed_buffer_	520
6.41.4.2 fixed_buffer_size_	520
6.41.4.3 fixed_buffer_used_size_	520
6.41.4.4 growable_buffer_	520
6.41.4.5 strm_	521
6.42 Класс http://TaskQueue	521
6.42.1 Подробное описание	522
6.42.2 Конструктор(ы)	522
6.42.2.1 TaskQueue()	522
6.42.2.2 ~TaskQueue()	522
6.42.3 Методы	522
6.42.3.1 enqueue()	522
6.42.3.2 on_idle()	523
6.42.3.3 shutdown()	523
6.43 Класс http://ThreadPool	523
6.43.1 Подробное описание	525
6.43.2 Конструктор(ы)	525
6.43.2.1 ThreadPool() [1/2]	525
6.43.2.2 ThreadPool() [2/2]	526
6.43.2.3 ~ThreadPool()	526
6.43.3 Методы	526
6.43.3.1 enqueue()	526
6.43.3.2 shutdown()	527
6.43.4 Друзья класса и относящимся к классу обозначения	527
6.43.4.1 worker	527
6.43.5 Данные класса	527
6.43.5.1 cond_	527
6.43.5.2 jobs_	527
6.43.5.3 max_queued_requests_	527
6.43.5.4 mutex_	527
6.43.5.5 shutdown_	528
6.43.5.6 threads_	528
6.44 Структура User	528
6.44.1 Подробное описание	529

6.44.2 Данные класса	529
6.44.2.1 id	529
6.44.2.2 password	529
6.44.2.3 username	529
6.45 Структура <code>httpplib::ThreadPool::worker</code>	530
6.45.1 Подробное описание	531
6.45.2 Конструктор(ы)	531
6.45.2.1 <code>worker()</code>	531
6.45.3 Методы	531
6.45.3.1 <code>operator()()</code>	531
6.45.4 Данные класса	532
6.45.4.1 <code>pool_</code>	532
7 Файлы	533
7.1 Файл <code>Base64URL.h</code>	533
7.2 <code>Base64URL.h</code>	534
7.3 Файл <code>BigInt.h</code>	534
7.4 <code>BigInt.h</code>	535
7.5 Файл <code>Database.h</code>	536
7.6 <code>Database.h</code>	537
7.7 Файл <code>httpplib.h</code>	537
7.7.1 Макросы	545
7.7.1.1 <code>CPPHTTPPLIB_CLIENT_MAX_TIMEOUT_MSECOND</code>	545
7.7.1.2 <code>CPPHTTPPLIB_CLIENT_READ_TIMEOUT_SECOND</code>	545
7.7.1.3 <code>CPPHTTPPLIB_CLIENT_READ_TIMEOUT_USECOND</code>	545
7.7.1.4 <code>CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_SECOND</code>	545
7.7.1.5 <code>CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_USECOND</code>	545
7.7.1.6 <code>CPPHTTPPLIB_COMPRESSION_BUFSIZ</code>	546
7.7.1.7 <code>CPPHTTPPLIB_CONNECTION_TIMEOUT_SECOND</code>	546
7.7.1.8 <code>CPPHTTPPLIB_CONNECTION_TIMEOUT_USECOND</code>	546
7.7.1.9 <code>CPPHTTPPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH</code>	546
7.7.1.10 <code>CPPHTTPPLIB_HEADER_MAX_LENGTH</code>	546
7.7.1.11 <code>CPPHTTPPLIB_IDLE_INTERVAL_SECOND</code>	546
7.7.1.12 <code>CPPHTTPPLIB_IDLE_INTERVAL_USECOND</code>	546
7.7.1.13 <code>CPPHTTPPLIB_IPV6_V6ONLY</code>	546
7.7.1.14 <code>CPPHTTPPLIB_KEEPALIVE_MAX_COUNT</code>	547
7.7.1.15 <code>CPPHTTPPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND</code>	547
7.7.1.16 <code>CPPHTTPPLIB_KEEPALIVE_TIMEOUT_SECOND</code>	547
7.7.1.17 <code>CPPHTTPPLIB_LISTEN_BACKLOG</code>	547
7.7.1.18 <code>CPPHTTPPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT</code>	547
7.7.1.19 <code>CPPHTTPPLIB_PAYLOAD_MAX_LENGTH</code>	547
7.7.1.20 <code>CPPHTTPPLIB_RANGE_MAX_COUNT</code>	547
7.7.1.21 <code>CPPHTTPPLIB_RECV_BUFSIZ</code>	547

7.7.1.22 CPPHTTPLIB_RECV_FLAGS	548
7.7.1.23 CPPHTTPLIB_REDIRECT_MAX_COUNT	548
7.7.1.24 CPPHTTPLIB_REQUEST_URI_MAX_LENGTH	548
7.7.1.25 CPPHTTPLIB_SEND_FLAGS	548
7.7.1.26 CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND	548
7.7.1.27 CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND	548
7.7.1.28 CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND	548
7.7.1.29 CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USECOND	548
7.7.1.30 CPPHTTPLIB_TCP_NODELAY	549
7.7.1.31 CPPHTTPLIB_THREAD_POOL_COUNT	549
7.7.1.32 CPPHTTPLIB_VERSION	549
7.7.1.33 INVALID_SOCKET	549
7.7.1.34 USE_IF2IP	549
7.7.2 Типы	549
7.7.2.1 socket_t	549
7.8 httplib.h	550
7.9 Файл HttpServer.h	670
7.10 HttpServer.h	671
7.11 Файл JWT.h	671
7.12 JWT.h	672
7.13 Файл KeyStorage.h	672
7.14 KeyStorage.h	673
7.15 Файл PasswordEncryptor.h	673
7.16 PasswordEncryptor.h	674
7.17 Файл RSA.h	674
7.18 RSA.h	675
7.19 Файл SHA256.h	676
7.20 SHA256.h	676
7.21 Файл Base64URL.cpp	677
7.21.1 Переменные	677
7.21.1.1 base64_chars	677
7.22 Base64URL.cpp	677
7.23 Файл BigInt.cpp	678
7.24 BigInt.cpp	679
7.25 Файл Database.cpp	683
7.25.1 Переменные	683
7.25.1.1 db	683
7.26 Database.cpp	683
7.27 Файл HttpServer.cpp	685
7.27.1 Функции	685
7.27.1.1 extractField()	685
7.28 HttpServer.cpp	686
7.29 Файл JWT.cpp	690

7.30 JWT.cpp	690
7.31 Файл KeyStorage.cpp	693
7.31.1 Переменные	694
7.31.1.1 PRIV_FILE	694
7.31.1.2 PUB_FILE	694
7.32 KeyStorage.cpp	694
7.33 Файл main.cpp	695
7.33.1 Функции	695
7.33.1.1 main()	695
7.34 main.cpp	696
7.35 Файл PasswordEncryptor.cpp	696
7.36 PasswordEncryptor.cpp	697
7.37 Файл RSA.cpp	697
7.37.1 Функции	698
7.37.1.1 is_prime()	698
7.37.1.2 modinv()	699
7.37.1.3 random_bigint()	700
7.38 RSA.cpp	700
7.39 Файл SHA256.cpp	702
7.40 SHA256.cpp	704
Предметный указатель	706

Глава 1

Алфавитный указатель пространств имен

1.1 Пространства имен

Полный список пространств имен.

anonymous_namespace{SHA256.cpp}	7
httpplib	12
httpplib::detail	28
httpplib::detail::case_ignore	125
httpplib::detail::fields	126
httpplib::detail::udl	134

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

Base64URL	136
BigInt	140
httpplib::Client	166
httpplib::ClientImpl	201
httpplib::detail::compressor	305
httpplib::detail::nocompressor	371
httpplib::detail::ContentProviderAdapter	307
httpplib::ContentReader	309
Database	314
httpplib::DataSink	320
httpplib::detail::decompressor	324
httpplib::detail::case_ignore::equal_to	326
httpplib::detail::FileStat	327
httpplib::detail::case_ignore::hash	329
HttpServer	331
JWT	336
KeyStorage	344
httpplib::detail::MatcherBase	348
httpplib::detail::PathParamsMatcher	374
httpplib::detail::RegexMatcher	379
httpplib::detail::mmap	349
httpplib::Server::MountPointEntry	355
httpplib::MultipartFormData	356
httpplib::detail::MultipartFormDataParser	358
httpplib::MultipartFormDataProvider	369
PasswordEncryptor	373
httpplib::Request	381
httpplib::Response	395
httpplib::Result	411
RSA	417
RSAPrivateKey	424
RSAPublicKey	426
httpplib::detail::scope_exit	427
httpplib::Server	431
SHA256	495

httpplib::ClientImpl::Socket	497
httpplib::Stream	509
httpplib::detail::BufferStream	160
httpplib::detail::SocketStream	498
httpplib::detail::stream_line_reader	515
std::streambuf	
httpplib::DataSink::data_sink_streambuf	311
httpplib::TaskQueue	521
httpplib::ThreadPool	523
User	528
httpplib::ThreadPool::worker	530

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

Base64URL	Класс для кодирования и декодирования строк в формате Base64URL	136
BigInt	Класс для работы с большими целыми числами произвольной длины (Big Integer)	140
httpplib::detail::BufferStream	httpplib::detail::BufferStream	160
httpplib::Client	httpplib::Client	166
httpplib::ClientImpl	httpplib::ClientImpl	201
httpplib::detail::compressor	httpplib::detail::compressor	305
httpplib::detail::ContentProviderAdapter	httpplib::detail::ContentProviderAdapter	307
httpplib::ContentReader	httpplib::ContentReader	309
httpplib::DataSink::data_sink_streambuf	httpplib::DataSink::data_sink_streambuf	311
Database	Класс-обёртка для работы с SQLite-базой данных	314
httpplib::DataSink	httpplib::DataSink	320
httpplib::detail::decompressor	httpplib::detail::decompressor	324
httpplib::detail::case_ignore::equal_to	httpplib::detail::case_ignore::equal_to	326
httpplib::detail::FileStat	httpplib::detail::FileStat	327
httpplib::detail::case_ignore::hash	httpplib::detail::case_ignore::hash	329
HttpServer	Класс для запуска HTTP-сервера авторизации	331
JWT	Класс, реализующий создание и проверку JSON Web Token (JWT)	336
KeyStorage	Класс для хранения и загрузки RSA-ключей на диск	344
httpplib::detail::MatcherBase	httpplib::detail::MatcherBase	348
httpplib::detail::mmap	httpplib::detail::mmap	349
httpplib::Server::MountPointEntry	httpplib::Server::MountPointEntry	355
httpplib::MultipartFormData	httpplib::MultipartFormData	356
httpplib::detail::MultipartFormDataParser	httpplib::detail::MultipartFormDataParser	358
httpplib::MultipartFormDataProvider	httpplib::MultipartFormDataProvider	369
httpplib::detail::nocompressor	httpplib::detail::nocompressor	371
PasswordEncryptor	Утилита для безопасного хеширования паролей	373
httpplib::detail::PathParamMatcher	httpplib::detail::PathParamMatcher	374
httpplib::detail::RegexMatcher	httpplib::detail::RegexMatcher	379

httpplib::Request	381
httpplib::Response	395
httpplib::Result	411
RSA	
Класс, реализующий основные операции алгоритма RSA	417
RSAPrivateKey	
Структура для хранения закрытого (приватного) ключа RSA	424
RSAPublicKey	
Структура для хранения открытого (публичного) ключа RSA	426
httpplib::detail::scope_exit	427
httpplib::Server	431
SHA256	
Реализация криптографического хеш-функции SHA-256	495
httpplib::ClientImpl::Socket	497
httpplib::detail::SocketStream	498
httpplib::Stream	509
httpplib::detail::stream_line_reader	515
httpplib::TaskQueue	521
httpplib::ThreadPool	523
User	
Структура, представляющая пользователя из базы данных	528
httpplib::ThreadPool::worker	530

Глава 4

Список файлов

4.1 Файлы

Полный список файлов.

Base64URL.h	533
BigInt.h	534
Database.h	536
httpplib.h	537
HttpServer.h	670
JWT.h	671
KeyStorage.h	672
PasswordEncryptor.h	673
RSA.h	674
SHA256.h	676
Base64URL.cpp	677
BigInt.cpp	678
Database.cpp	683
HttpServer.cpp	685
JWT.cpp	690
KeyStorage.cpp	693
main.cpp	695
PasswordEncryptor.cpp	696
RSA.cpp	697
SHA256.cpp	702

Глава 5

Пространства имен

5.1 Пространство имен anonymous_namespace{SHA256.cpp}

Функции

- `uint32_t rotr (uint32_t x, uint32_t n)`
Побитовый циклический сдвиг вправо.
- `uint32_t ch (uint32_t x, uint32_t y, uint32_t z)`
Вычисляет функцию выбора: выбирает `y`, если `x`, иначе `z`.
- `uint32_t maj (uint32_t x, uint32_t y, uint32_t z)`
Мажоритарная функция: возвращает значение, встречающееся чаще всего среди `x`, `y`, `z`.
- `uint32_t big_sigma0 (uint32_t x)`
Функция 0 из SHA-256: используется в расширении блока.
- `uint32_t big_sigma1 (uint32_t x)`
Функция 1 из SHA-256: используется в расширении блока.
- `uint32_t small_sigma0 (uint32_t x)`
Функция 0 из SHA-256: используется при генерации `w[16..63]`.
- `uint32_t small_sigma1 (uint32_t x)`
Функция 1 из SHA-256: используется при генерации `w[16..63]`.
- `std::vector< uint8_t > pad (const std::string &input)`
Дополняет вход до кратного 512 бит (64 байта).
- `uint32_t to_uint32 (const uint8_t *bytes)`
Преобразует 4 байта в 32-битное целое число (big-endian).

Переменные

- `const std::array< uint32_t, 64 > k`
Константы, используемые в каждом из 64 раундов SHA-256.

5.1.1 Функции

5.1.1.1 big_sigma0()

```
uint32_t anonymous_namespace{SHA256.cpp}::big_sigma0 (
    uint32_t x) [inline]
```

Функция 0 из SHA-256: используется в расширении блока.

См. определение в файле [SHA256.cpp](#) строка 63

```
00063     {
00064         return rotr(x, 2) ^ rotr(x, 13) ^ rotr(x, 22);
00065     }
```

Граф вызовов:



5.1.1.2 big_sigma1()

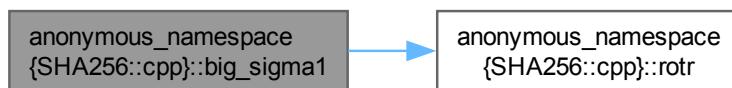
```
uint32_t anonymous_namespace{SHA256.cpp}::big_sigma1 (
    uint32_t x) [inline]
```

Функция 1 из SHA-256: используется в расширении блока.

См. определение в файле [SHA256.cpp](#) строка 70

```
00070     {
00071         return rotr(x, 6) ^ rotr(x, 11) ^ rotr(x, 25);
00072     }
```

Граф вызовов:



5.1.1.3 ch()

```
uint32_t anonymous_namespace{SHA256.cpp}::ch (
    uint32_t x,
    uint32_t y,
    uint32_t z) [inline]
```

Вычисляет функцию выбора: выбирает у, если x, иначе z.

Формально: (x AND y) XOR (NOT x AND z)

См. определение в файле [SHA256.cpp](#) строка 47

```
00047
00048     return (x & y) ^ (~x & z);
00049 }
```

5.1.1.4 maj()

```
uint32_t anonymous_namespace{SHA256.cpp}::maj (
    uint32_t x,
    uint32_t y,
    uint32_t z) [inline]
```

Мажоритарная функция: возвращает значение, встречающееся чаще всего среди x, y, z.

Формально: (x AND y) XOR (x AND z) XOR (y AND z)

См. определение в файле [SHA256.cpp](#) строка 56

```
00056
00057     return (x & y) ^ (x & z) ^ (y & z);
00058 }
```

5.1.1.5 pad()

```
std::vector< uint8_t > anonymous_namespace{SHA256.cpp}::pad (
    const std::string & input)
```

Дополняет вход до кратного 512 бит (64 байта).

Алгоритм добавляет:

- Один бит 1
- 0...0 (до выравнивания)
- Длину исходного сообщения в битах (64 бита)

Аргументы

input	Исходная строка
-------	-----------------

Возвращает

Вектор байт после паддинга

См. определение в файле [SHA256.cpp](#) строка 99

```
00099
00100     size_t original_length = input.size();
00101     uint64_t bit_length = original_length * 8;
00102
00103     std::vector<uint8_t> padded(input.begin(), input.end());
00104     padded.push_back(0x80);
00105     while ((padded.size() + 8) % 64 != 0) padded.push_back(0x00);
00106     for (int i = 7; i >= 0; --i)
00107         padded.push_back((bit_length >> (i * 8)) & 0xFF);
00108
00109     return padded;
00110 }
```

5.1.1.6 rot()

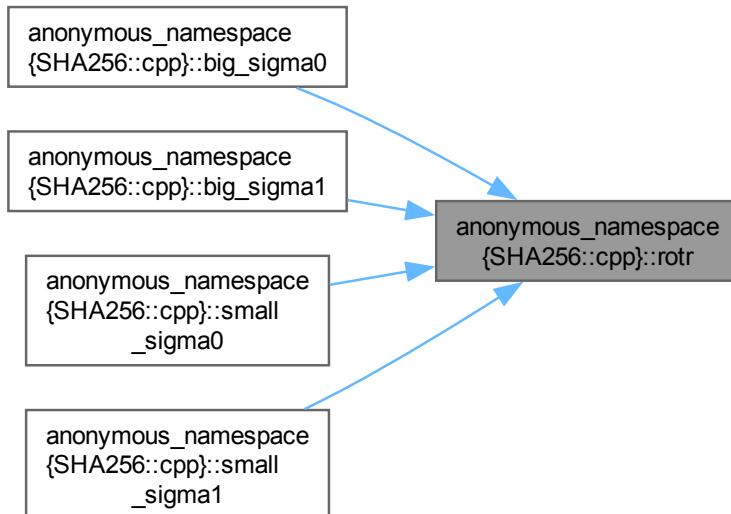
```
uint32_t anonymous_namespace{SHA256.cpp}::rot (
    uint32_t x,
    uint32_t n) [inline]
```

Побитовый циклический сдвиг вправо.

См. определение в файле [SHA256.cpp](#) строка 38

```
00038     {
00039         return (x >> n) | (x << (32 - n));
00040     }
```

Граф вызова функции:



5.1.1.7 small_sigma0()

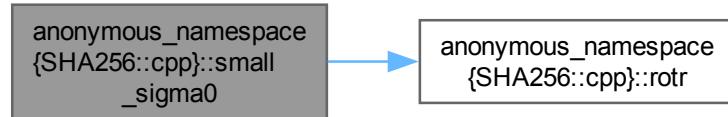
```
uint32_t anonymous_namespace{SHA256.cpp}::small_sigma0 (
    uint32_t x) [inline]
```

Функция 0 из SHA-256: используется при генерации w[16..63].

См. определение в файле [SHA256.cpp](#) строка 77

```
00077     {
00078         return rot(x, 7) ^ rot(x, 18) ^ (x >> 3);
00079     }
```

Граф вызовов:



5.1.1.8 small_sigma1()

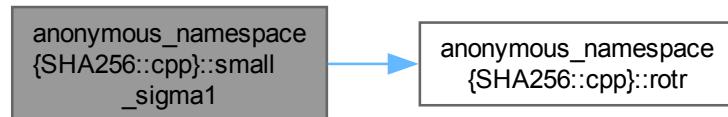
```
uint32_t anonymous_namespace{SHA256.cpp}::small_sigma1 (
    uint32_t x) [inline]
```

Функция 1 из SHA-256: используется при генерации w[16..63].

См. определение в файле [SHA256.cpp](#) строка 84

```
00084     {
00085         return rotr(x, 17) ^ rotr(x, 19) ^ (x >> 10);
00086     }
```

Граф вызовов:



5.1.1.9 to_uint32()

```
uint32_t anonymous_namespace{SHA256.cpp}::to_uint32 (
    const uint8_t * bytes)
```

Преобразует 4 байта в 32-битное целое число (big-endian).

См. определение в файле [SHA256.cpp](#) строка 115

```
00115     {
00116         return (bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3];
00117     }
```

5.1.2 Переменные

5.1.2.1 k

```
const std::array<uint32_t, 64> anonymous_namespace{SHA256.cpp}::k
```

Инициализатор

```
= {
    0x428a2f98, 0x71374491, 0xb5c0fbef, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0xfc19dc6, 0x240ca1cc,
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
    0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
    0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
    0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
    0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
    0x90beffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
}
```

Константы, используемые в каждом из 64 раундов SHA-256.

Эти значения представляют собой первые 32 бита дробных частей кубических корней первых 64 простых чисел.

См. определение в файле [SHA256.cpp](#) строка 16

```
00016    {
00017        0x428a2f98, 0x71374491, 0xb5c0fbef, 0xe9b5dba5,
00018        0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
00019        0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
00020        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
00021        0xe49b69c1, 0xefbe4786, 0xfc19dc6, 0x240ca1cc,
00022        0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
00023        0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
00024        0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
00025        0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
00026        0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
00027        0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
00028        0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
00029        0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
00030        0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
00031        0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
00032        0x90beffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
00033    };
```

5.2 Пространство имен `httpplib`

Пространства имен

- namespace [detail](#)

Классы

- class [Client](#)
- class [ClientImpl](#)
- class [ContentReader](#)
- class [DataSink](#)
- struct [MultipartFormData](#)
- struct [MultipartFormDataProvider](#)
- struct [Request](#)
- struct [Response](#)
- class [Result](#)
- class [Server](#)
- class [Stream](#)
- class [TaskQueue](#)
- class [ThreadPool](#)

Определения типов

- using `Headers`
- using `Params` = `std::multimap<std::string, std::string>`
- using `Match` = `std::smatch`
- using `Progress` = `std::function<bool(uint64_t current, uint64_t total)>`
- using `ResponseHandler` = `std::function<bool(const Response &response)>`
- using `MultipartFormDataItems` = `std::vector<MultipartFormData>`
- using `MultipartFormDataMap` = `std::multimap<std::string, MultipartFormData>`
- using `ContentProvider`
- using `ContentProviderWithoutLength`
- using `ContentProviderResourceReleaser` = `std::function<void(bool success)>`
- using `MultipartFormDataProviderItems` = `std::vector<MultipartFormDataProvider>`
- using `ContentReceiverWithProgress`
- using `ContentReceiver`
- using `MultipartContentHeader`
- using `Range` = `std::pair<ssize_t, ssize_t>`
- using `Ranges` = `std::vector<Range>`
- using `Logger` = `std::function<void(const Request &, const Response &)>`
- using `SocketOptions` = `std::function<void(socket_t sock)>`

Перечисления

- enum `SSLVerifierResponse` { `NoDecisionMade` , `CertificateAccepted` , `CertificateRejected` }
- enum `StatusCode` {
 `Continue_100` = 100 , `SwitchingProtocol_101` = 101 , `Processing_102` = 102 , `EarlyHints_103` = 103 ,
 `OK_200` = 200 , `Created_201` = 201 , `Accepted_202` = 202 , `NonAuthoritativeInformation_203` = 203 ,
 `NoContent_204` = 204 , `ResetContent_205` = 205 , `PartialContent_206` = 206 , `MultiStatus_207` = 207 ,
 `AlreadyReported_208` = 208 , `IMUsed_226` = 226 , `MultipleChoices_300` = 300 , `MovedPermanently_301` = 301 ,
 `Found_302` = 302 , `SeeOther_303` = 303 , `NotModified_304` = 304 , `UseProxy_305` = 305 ,
 `unused_306` = 306 , `TemporaryRedirect_307` = 307 , `PermanentRedirect_308` = 308 ,
 `BadRequest_400` = 400 ,
 `Unauthorized_401` = 401 , `PaymentRequired_402` = 402 , `Forbidden_403` = 403 , `NotFound_404` = 404 ,
 `MethodNotAllowed_405` = 405 , `NotAcceptable_406` = 406 , `ProxyAuthenticationRequired_407` = 407 ,
 `RequestTimeout_408` = 408 ,
 `Conflict_409` = 409 , `Gone_410` = 410 , `LengthRequired_411` = 411 , `PreconditionFailed_412` = 412 ,
 `PayloadTooLarge_413` = 413 , `UriTooLong_414` = 414 , `UnsupportedMediaType_415` = 415 ,
 `RangeNotSatisfiable_416` = 416 ,
 `ExpectationFailed_417` = 417 , `ImATeapot_418` = 418 , `MisdirectedRequest_421` = 421 ,
 `UnprocessableContent_422` = 422 ,
 `Locked_423` = 423 , `FailedDependency_424` = 424 , `TooEarly_425` = 425 , `UpgradeRequired_426` = 426 ,
 `PreconditionRequired_428` = 428 , `TooManyRequests_429` = 429 , `RequestHeaderFieldsTooLarge_431` = 431 ,
 `UnavailableForLegalReasons_451` = 451 ,
 `InternalServerError_500` = 500 , `NotImplemented_501` = 501 , `BadGateway_502` = 502 ,
 `ServiceUnavailable_503` = 503 ,
 `GatewayTimeout_504` = 504 , `HttpVersionNotSupported_505` = 505 , `VariantAlsoNegotiates_506` = 506 ,
 `InsufficientStorage_507` = 507 ,
 `LoopDetected_508` = 508 , `NotExtended_510` = 510 , `NetworkAuthenticationRequired_511` = 511 }

- enum class **Error** {
 Success = 0 , Unknown , Connection , BindIPAddress ,
 Read , Write , ExceedRedirectCount , Canceled ,
 SSLConnection , SSLLoadingCerts , SSLServerVerification , SSLServerHostnameVerification ,
 UnsupportedMultipartBoundaryChars , Compression , ConnectionTimeout , ProxyConnection ,
 SSLPeerCouldBeClosed_ }

Функции

- void **default_socket_options** (**socket_t** sock)
- const char * **status_message** (int status)
- std::string **get_bearer_token_auth** (const **Request** &req)
- std::string **to_string** (**Error** error)
- std::ostream & **operator<<** (std::ostream &os, const **Error** &obj)
- std::string **hosted_at** (const std::string &hostname)
- void **hosted_at** (const std::string &hostname, std::vector< std::string > &addrs)
- std::string **append_query_params** (const std::string &path, const **Params** ¶ms)
- std::pair< std::string, std::string > **make_range_header** (const **Ranges** &ranges)
- std::pair< std::string, std::string > **make_basic_authentication_header** (const std::string &username, const std::string &password, bool is_proxy=false)
- std::pair< std::string, std::string > **make_bearer_token_authentication_header** (const std::string &token, bool is_proxy=false)

5.2.1 Типы

5.2.1.1 ContentProvider

using **httpplib::ContentProvider**

Инициализатор

std::function<bool(size_t offset, size_t length, **DataSink** &sink)>

См. определение в файле **httpplib.h** строка 575

5.2.1.2 ContentProviderResourceReleaser

using **httpplib::ContentProviderResourceReleaser** = std::function<void(bool success)>

См. определение в файле **httpplib.h** строка 581

5.2.1.3 ContentProviderWithoutLength

using **httpplib::ContentProviderWithoutLength**

Инициализатор

std::function<bool(size_t offset, **DataSink** &sink)>

См. определение в файле **httpplib.h** строка 578

5.2.1.4 ContentReceiver

```
using httpplib::ContentReceiver
```

Инициализатор

```
std::function<bool(const char *data, size_t data_length)>
```

См. определение в файле `httpplib.h` строка [595](#)

5.2.1.5 ContentReceiverWithProgress

```
using httpplib::ContentReceiverWithProgress
```

Инициализатор

```
std::function<bool(const char *data, size_t data_length, uint64_t offset,
                   uint64_t total_length)>
```

См. определение в файле `httpplib.h` строка [591](#)

5.2.1.6 Headers

```
using httpplib::Headers
```

Инициализатор

```
std::unordered_multimap<std::string, std::string, detail::case_ignore::hash,
                      detail::case_ignore::equal_to>
```

См. определение в файле `httpplib.h` строка [521](#)

5.2.1.7 Logger

```
using httpplib::Logger = std::function<void(const Request &, const Response &)>
```

См. определение в файле `httpplib.h` строка [857](#)

5.2.1.8 Match

```
using httpplib::Match = std::smatch
```

См. определение в файле `httpplib.h` строка [526](#)

5.2.1.9 MultipartContentHeader

```
using httpplib::MultipartContentHeader
```

Инициализатор

```
std::function<bool(const MultipartFormData &file)>
```

См. определение в файле `httpplib.h` строка [598](#)

5.2.1.10 `MultipartFormDataItems`

```
using httpplib::MultipartFormDataItems = std::vector<MultipartFormData>
```

См. определение в файле `httpplib.h` строка [539](#)

5.2.1.11 `MultipartFormDataMap`

```
using httpplib::MultipartFormDataMap = std::multimap<std::string, MultipartFormData>
```

См. определение в файле `httpplib.h` строка [540](#)

5.2.1.12 `MultipartFormDataProviderItems`

```
using httpplib::MultipartFormDataProviderItems = std::vector<MultipartFormDataProvider>
```

См. определение в файле `httpplib.h` строка [589](#)

5.2.1.13 `Params`

```
using httpplib::Params = std::multimap<std::string, std::string>
```

См. определение в файле `httpplib.h` строка [525](#)

5.2.1.14 `Progress`

```
using httpplib::Progress = std::function<bool(uint64_t current, uint64_t total)>
```

См. определение в файле `httpplib.h` строка [528](#)

5.2.1.15 `Range`

```
using httpplib::Range = std::pair<ssize_t, ssize_t>
```

См. определение в файле `httpplib.h` строка [624](#)

5.2.1.16 `Ranges`

```
using httpplib::Ranges = std::vector<Range>
```

См. определение в файле `httpplib.h` строка [625](#)

5.2.1.17 `ResponseHandler`

```
using httpplib::ResponseHandler = std::function<bool(const Response &response)>
```

См. определение в файле `httpplib.h` строка [531](#)

5.2.1.18 `SocketOptions`

```
using httpplib::SocketOptions = std::function<void(socket_t sock)>
```

См. определение в файле `httpplib.h` строка [859](#)

5.2.2 Перечисления

5.2.2.1 `Error`

```
enum class httpplib::Error [strong]
```

Элементы перечислений

Success	
Unknown	
Connection	
BindIPAddress	
Read	
Write	
ExceedRedirectCount	
Canceled	
SSLConnection	
SSLLoadingCerts	
SSLServerVerification	
SSLServerHostnameVerification	
UnsupportedMultipartBoundaryChars	
Compression	
ConnectionTimeout	
ProxyConnection	
SSLPeerCouldBeClosed_	

См. определение в файле `httpplib.h` строка 1162

```
01162     {
01163     Success = 0,
01164     Unknown,
01165     Connection,
01166     BindIPAddress,
01167     Read,
01168     Write,
01169     ExceedRedirectCount,
01170     Canceled,
01171     SSLConnection,
01172     SSLLoadingCerts,
01173     SSLServerVerification,
01174     SSLServerHostnameVerification,
01175     UnsupportedMultipartBoundaryChars,
01176     Compression,
01177     ConnectionTimeout,
01178     ProxyConnection,
01179
01180 // For internal use only
01181     SSLPeerCouldBeClosed_,
01182 };
```

5.2.2.2 SSLVerifierResponse

enum `httpplib::SSLVerifierResponse`

Элементы перечислений

NoDecisionMade	
CertificateAccepted	
CertificateRejected	

См. определение в файле `httpplib.h` строка 437

```
00437     {
00438 // no decision has been made, use the built-in certificate verifier
00439     NoDecisionMade,
00440 // connection certificate is verified and accepted
00441     CertificateAccepted,
00442 // connection certificate was processed but is rejected
00443     CertificateRejected
00444 };
```

5.2.2.3 StatusCode

enum [httplib::StatusCode](#)

Элементы перечислений

Continue_100	
SwitchingProtocol_101	
Processing_102	
EarlyHints_103	
OK_200	
Created_201	
Accepted_202	
NonAuthoritativeInformation_203	
NoContent_204	
ResetContent_205	
PartialContent_206	
MultiStatus_207	
AlreadyReported_208	
IMUsed_226	
MultipleChoices_300	
MovedPermanently_301	
Found_302	
SeeOther_303	
NotModified_304	
UseProxy_305	
unused_306	
TemporaryRedirect_307	
PermanentRedirect_308	
BadRequest_400	
Unauthorized_401	
PaymentRequired_402	
Forbidden_403	
NotFound_404	
MethodNotAllowed_405	
NotAcceptable_406	
ProxyAuthenticationRequired_407	
RequestTimeout_408	
Conflict_409	
Gone_410	
LengthRequired_411	
PreconditionFailed_412	
PayloadTooLarge_413	
UriTooLong_414	
UnsupportedMediaType_415	
RangeNotSatisfiable_416	
ExpectationFailed_417	

Элементы перечислений

IMATeapot_418
MisdirectedRequest_421
UnprocessableContent_422
Locked_423
FailedDependency_424
TooEarly_425
UpgradeRequired_426
PreconditionRequired_428
TooManyRequests_429
RequestHeaderFieldsTooLarge_431
UnavailableForLegalReasons_451
InternalServerError_500
NotImplemented_501
BadGateway_502
ServiceUnavailable_503
GatewayTimeout_504
HttpVersionNotSupported_505
VariantAlsoNegotiates_506
InsufficientStorage_507
LoopDetected_508
NotExtended_510
NetworkAuthenticationRequired_511

См. определение в файле [httpplib.h](#) строка 446

```

00446 {
00447 // Information responses
00448 Continue_100 = 100,
00449 SwitchingProtocol_101 = 101,
00450 Processing_102 = 102,
00451 EarlyHints_103 = 103,
00452
00453 // Successful responses
00454 OK_200 = 200,
00455 Created_201 = 201,
00456 Accepted_202 = 202,
00457 NonAuthoritativeInformation_203 = 203,
00458 NoContent_204 = 204,
00459 ResetContent_205 = 205,
00460 PartialContent_206 = 206,
00461 MultiStatus_207 = 207,
00462 AlreadyReported_208 = 208,
00463 IMUsed_226 = 226,
00464
00465 // Redirection messages
00466 MultipleChoices_300 = 300,
00467 MovedPermanently_301 = 301,
00468 Found_302 = 302,
00469 SeeOther_303 = 303,
00470 NotModified_304 = 304,
00471 UseProxy_305 = 305,
00472 unused_306 = 306,
00473 TemporaryRedirect_307 = 307,
00474 PermanentRedirect_308 = 308,
00475
00476 // Client error responses
00477 BadRequest_400 = 400,
00478 Unauthorized_401 = 401,
00479 PaymentRequired_402 = 402,
00480 Forbidden_403 = 403,
00481 NotFound_404 = 404,
00482 MethodNotAllowed_405 = 405,

```

```

00483 NotAcceptable_406 = 406,
00484 ProxyAuthenticationRequired_407 = 407,
00485 RequestTimeout_408 = 408,
00486 Conflict_409 = 409,
00487 Gone_410 = 410,
00488 LengthRequired_411 = 411,
00489 PreconditionFailed_412 = 412,
00490 PayloadTooLarge_413 = 413,
00491 UriTooLong_414 = 414,
00492 UnsupportedMediaType_415 = 415,
00493 RangeNotSatisfiable_416 = 416,
00494 ExpectationFailed_417 = 417,
00495 ImATeapot_418 = 418,
00496 MisdirectedRequest_421 = 421,
00497 UnprocessableContent_422 = 422,
00498 Locked_423 = 423,
00499 FailedDependency_424 = 424,
00500 TooEarly_425 = 425,
00501 UpgradeRequired_426 = 426,
00502 PreconditionRequired_428 = 428,
00503 TooManyRequests_429 = 429,
00504 RequestHeaderFieldsTooLarge_431 = 431,
00505 UnavailableForLegalReasons_451 = 451,
00506
00507 // Server error responses
00508 InternalServerError_500 = 500,
00509 NotImplemented_501 = 501,
00510 BadGateway_502 = 502,
00511 ServiceUnavailable_503 = 503,
00512 GatewayTimeout_504 = 504,
00513 HttpVersionNotSupported_505 = 505,
00514 VariantAlsoNegotiates_506 = 506,
00515 InsufficientStorage_507 = 507,
00516 LoopDetected_508 = 508,
00517 NotExtended_510 = 510,
00518 NetworkAuthenticationRequired_511 = 511,
00519 };

```

5.2.3 Функции

5.2.3.1 `append_query_params()`

```

std::string httpplib::append_query_params (
    const std::string & path,
    const Params & params) [inline]

```

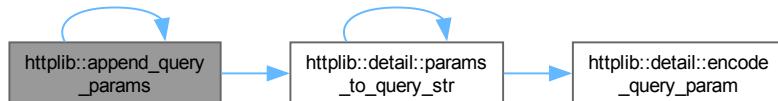
См. определение в файле `httpplib.h` строка 5817

```

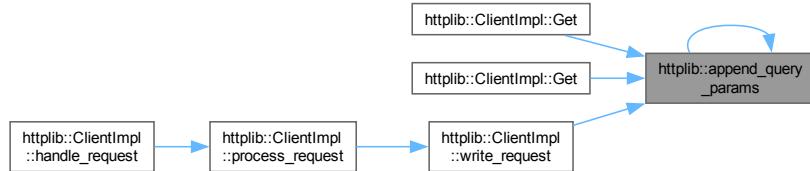
05818 {
05819     std::string path_with_query = path;
05820     thread_local const std::regex re("[^?]+\\?.*");
05821     auto delm = std::regex_match(path, re) ? '&' : '?';
05822     path_with_query += delm + detail::params_to_query_str(params);
05823     return path_with_query;
05824 }

```

Граф вызовов:



Граф вызова функции:



5.2.3.2 `default_socket_options()`

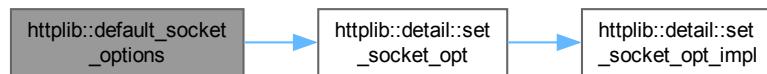
```
void httpplib::default_socket_options (
    socket_t sock) [inline]
```

См. определение в файле `httpplib.h` строка 2135

```

02135     {
02136     detail::set_socket_opt(sock, SOL_SOCKET,
02137     #ifdef SO_REUSEPORT
02138             SO_REUSEPORT,
02139     #else
02140             SO_REUSEADDR,
02141     #endif
02142             1);
02143 }
```

Граф вызовов:



5.2.3.3 `get_bearer_token_auth()`

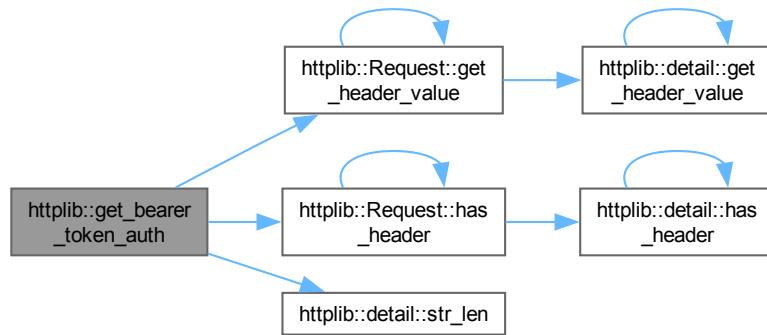
```
std::string httpplib::get_bearer_token_auth (
    const Request & req) [inline]
```

См. определение в файле `httpplib.h` строка 2221

```

02221     {
02222     if (req.has_header("Authorization")) {
02223         constexpr auto bearer_header_prefix_len = detail::str_len("Bearer ");
02224         return req.get_header_value("Authorization")
02225             .substr(bearer_header_prefix_len);
02226     }
02227     return "";
02228 }
```

Граф вызовов:



5.2.3.4 `hosted_at()` [1/2]

```
std::string httpplib::hosted_at (
    const std::string & hostname) [inline]
```

См. определение в файле [httpplib.h](#) строка 5780

```
05780
05781     std::vector<std::string> addrs;
05782     hosted_at(hostname, addrs);
05783     if (addrs.empty()) { return std::string(); }
05784     return addrs[0];
05785 }
```

Граф вызовов:



Граф вызова функции:



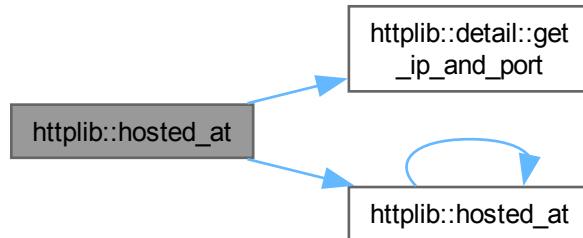
5.2.3.5 `hosted_at()` [2/2]

```
void httpplib::hosted_at (
    const std::string & hostname,
    std::vector< std::string > & addrs) [inline]
```

См. определение в файле `httpplib.h` строка 5787

```
05788                                     {
05789     struct addrinfo hints;
05790     struct addrinfo *result;
05791
05792     memset(&hints, 0, sizeof(struct addrinfo));
05793     hints.ai_family = AF_UNSPEC;
05794     hints.ai_socktype = SOCK_STREAM;
05795     hints.ai_protocol = 0;
05796
05797     if (getaddrinfo(hostname.c_str(), nullptr, &hints, &result)) {
05798 #if defined __linux__ && !defined __ANDROID__
05799         res_init();
05800 #endif
05801     return;
05802 }
05803 auto se = detail::scope_exit([&] { freeaddrinfo(result); });
05804
05805 for (auto rp = result; rp; rp = rp->ai_next) {
05806     const auto &addr =
05807         *reinterpret_cast<struct sockaddr_storage *>(rp->ai_addr);
05808     std::string ip;
05809     auto dummy = -1;
05810     if (detail::get_ip_and_port(addr, sizeof(struct sockaddr_storage), ip,
05811                                 dummy)) {
05812         addrs.push_back(ip);
05813     }
05814 }
05815 }
```

Граф вызовов:

5.2.3.6 `make_basic_authentication_header()`

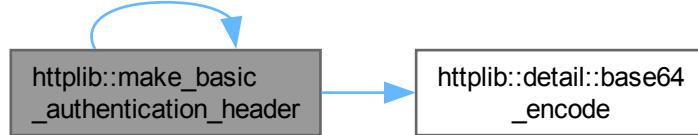
```
std::pair< std::string, std::string > httpplib::make_basic_authentication_header (
    const std::string & username,
    const std::string & password,
    bool is_proxy = false) [inline]
```

См. определение в файле `httpplib.h` строка 5842

```
05843                                     {
05844     auto field = "Basic " + detail::base64_encode(username + ":" + password);
05845     auto key = is_proxy ? "Proxy-Authorization" : "Authorization";
```

```
05846 return std::make_pair(key, std::move(field));
05847 }
```

Граф вызовов:



Граф вызова функции:



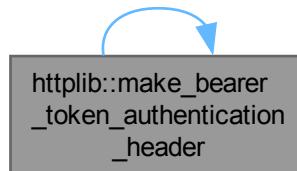
5.2.3.7 make_bearer_token_authentication_header()

```
std::pair< std::string, std::string > httplib::make_bearer_token_authentication_header (
    const std::string & token,
    bool is_proxy = false) [inline]
```

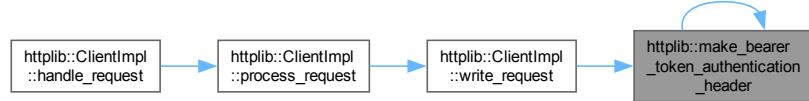
См. определение в файле [httplib.h](#) строка 5850

```
05851 {
05852     auto field = "Bearer " + token;
05853     auto key = is_proxy ? "Proxy-Authorization" : "Authorization";
05854     return std::make_pair(key, std::move(field));
05855 }
```

Граф вызовов:



Граф вызова функции:



5.2.3.8 make_range_header()

```
std::pair< std::string, std::string > httpplib::make_range_header (
    const Ranges & ranges) [inline]
```

См. определение в файле [httpplib.h](#) строка 5828

```

05828     {
05829     std::string field = "bytes=";
05830     auto i = 0;
05831     for (const auto &r : ranges) {
05832         if (i != 0) { field += ", ";}
05833         if (r.first != -1) { field += std::to_string(r.first); }
05834         field += "-";
05835         if (r.second != -1) { field += std::to_string(r.second); }
05836         i++;
05837     }
05838     return std::make_pair("Range", std::move(field));
05839 }
```

Граф вызовов:



Граф вызова функции:



5.2.3.9 `operator<<()`

```
std::ostream & httpplib::operator<< (
    std::ostream & os,
    const Error & obj) [inline]
```

См. определение в файле `httpplib.h` строка 2280

```
02280                                     {
02281     os << to_string(obj);
02282     os << " (" << static_cast<std::underlying_type<Error>::type>(obj) << ')';
02283     return os;
02284 }
```

Граф вызовов:



5.2.3.10 `status_message()`

```
const char * httpplib::status_message (
    int status) [inline]
```

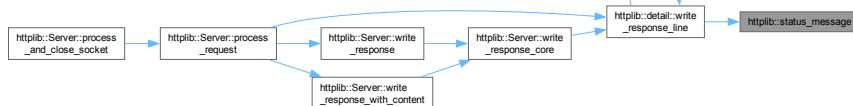
См. определение в файле `httpplib.h` строка 2145

```
02145                                     {
02146     switch (status) {
02147         case StatusCode::Continue_100: return "Continue";
02148         case StatusCode::SwitchingProtocol_101: return "Switching Protocol";
02149         case StatusCode::Processing_102: return "Processing";
02150         case StatusCode::EarlyHints_103: return "Early Hints";
02151         case StatusCode::OK_200: return "OK";
02152         case StatusCode::Created_201: return "Created";
02153         case StatusCode::Accepted_202: return "Accepted";
02154         case StatusCode::NonAuthoritativeInformation_203:
02155             return "Non-Authoritative Information";
02156         case StatusCode::NoContent_204: return "No Content";
02157         case StatusCode::ResetContent_205: return "Reset Content";
02158         case StatusCode::PartialContent_206: return "Partial Content";
02159         case StatusCode::MultiStatus_207: return "Multi-Status";
02160         case StatusCode::AlreadyReported_208: return "Already Reported";
02161         case StatusCode::IMUsed_226: return "IM Used";
02162         case StatusCode::MultipleChoices_300: return "Multiple Choices";
02163         case StatusCode::MovedPermanently_301: return "Moved Permanently";
02164         case StatusCode::Found_302: return "Found";
02165         case StatusCode::SeeOther_303: return "See Other";
02166         case StatusCode::NotModified_304: return "Not Modified";
02167         case StatusCode::UseProxy_305: return "Use Proxy";
02168         case StatusCode::unused_306: return "unused";
02169         case StatusCode::TemporaryRedirect_307: return "Temporary Redirect";
02170         case StatusCode::PermanentRedirect_308: return "Permanent Redirect";
02171         case StatusCode::BadRequest_400: return "Bad Request";
02172         case StatusCode::Unauthorized_401: return "Unauthorized";
02173         case StatusCode::PaymentRequired_402: return "Payment Required";
02174         case StatusCode::Forbidden_403: return "Forbidden";
02175         case StatusCode::NotFound_404: return "Not Found";
02176         case StatusCode::MethodNotAllowed_405: return "Method Not Allowed";
02177         case StatusCode::NotAcceptable_406: return "Not Acceptable";
02178         case StatusCode::ProxyAuthenticationRequired_407:
02179             return "Proxy Authentication Required";
02180         case StatusCode::RequestTimeout_408: return "Request Timeout";
02181         case StatusCode::Conflict_409: return "Conflict";
02182         case StatusCode::Gone_410: return "Gone";
    }
```

```

02183 case StatusCode::LengthRequired_411: return "Length Required";
02184 case StatusCode::PreconditionFailed_412: return "Precondition Failed";
02185 case StatusCode::PayloadTooLarge_413: return "Payload Too Large";
02186 case StatusCode::UriTooLong_414: return "URI Too Long";
02187 case StatusCode::UnsupportedMediaType_415: return "Unsupported Media Type";
02188 case StatusCode::RangeNotSatisfiable_416: return "Range Not Satisfiable";
02189 case StatusCode::ExpectationFailed_417: return "Expectation Failed";
02190 case StatusCode::ImATeapot_418: return "I'm a teapot";
02191 case StatusCode::MisdirectedRequest_421: return "Misdirected Request";
02192 case StatusCode::UnprocessableContent_422: return "Unprocessable Content";
02193 case StatusCode::Locked_423: return "Locked";
02194 case StatusCode::FailedDependency_424: return "Failed Dependency";
02195 case StatusCode::TooEarly_425: return "Too Early";
02196 case StatusCode::UpgradeRequired_426: return "Upgrade Required";
02197 case StatusCode::PreconditionRequired_428: return "Precondition Required";
02198 case StatusCode::TooManyRequests_429: return "Too Many Requests";
02199 case StatusCode::RequestHeaderFieldsTooLarge_431:
02200     return "Request Header Fields Too Large";
02201 case StatusCode::UnavailableForLegalReasons_451:
02202     return "Unavailable For Legal Reasons";
02203 case StatusCode::NotImplemented_501: return "Not Implemented";
02204 case StatusCode::BadGateway_502: return "Bad Gateway";
02205 case StatusCode::ServiceUnavailable_503: return "Service Unavailable";
02206 case StatusCode::GatewayTimeout_504: return "Gateway Timeout";
02207 case StatusCode::HttpVersionNotSupported_505:
02208     return "HTTP Version Not Supported";
02209 case StatusCode::VariantAlsoNegotiates_506: return "Variant Also Negotiates";
02210 case StatusCode::InsufficientStorage_507: return "Insufficient Storage";
02211 case StatusCode::LoopDetected_508: return "Loop Detected";
02212 case StatusCode::NotExtended_510: return "Not Extended";
02213 case StatusCode::NetworkAuthenticationRequired_511:
02214     return "Network Authentication Required";
02215
02216 default:
02217 case StatusCode::InternalServerError_500: return "Internal Server Error";
02218 }
02219 }
```

Граф вызова функции:



5.2.3.11 to_string()

```
std::string httpplib::to_string (
    Error error) [inline]
```

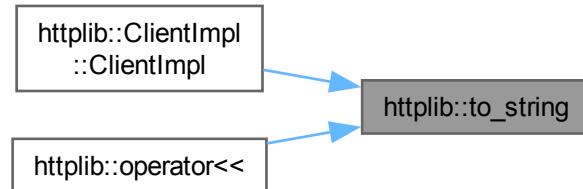
См. определение в файле [httpplib.h](#) строка 2254

```

02254 {
02255     switch (error) {
02256     case Error::Success: return "Success (no error)";
02257     case Error::Connection: return "Could not establish connection";
02258     case Error::BindIPAddress: return "Failed to bind IP address";
02259     case Error::Read: return "Failed to read connection";
02260     case Error::Write: return "Failed to write connection";
02261     case Error::ExceedRedirectCount: return "Maximum redirect count exceeded";
02262     case Error::Canceled: return "Connection handling canceled";
02263     case Error::SSLConnection: return "SSL connection failed";
02264     case Error::SSLLoadingCerts: return "SSL certificate loading failed";
02265     case Error::SSLServerVerification: return "SSL server verification failed";
02266     case Error::SSLServerHostnameVerification:
02267         return "SSL server hostname verification failed";
02268     case Error::UnsupportedMultipartBoundaryChars:
02269         return "Unsupported HTTP multipart boundary characters";
02270     case Error::Compression: return "Compression failed";
02271     case Error::ConnectionTimeout: return "Connection timed out";
02272     case Error::ProxyConnection: return "Proxy connection failed";
02273     case Error::Unknown: return "Unknown";
```

```
02274     default: break;
02275   }
02276
02277   return "Invalid";
02278 }
```

Граф вызова функции:



5.3 Пространство имен `httpplib::detail`

Пространства имен

- namespace `case_ignore`
- namespace `fields`
- namespace `ndl`

Классы

- class `BufferStream`
- class `compressor`
- class `ContentProviderAdapter`
- class `decompressor`
- struct `FileStat`
- class `MatcherBase`
- class `mmap`
- class `MultipartFormDataParser`
- class `nocompressor`
- class `PathParamMatcher`
- class `RegexMatcher`
- struct `scope_exit`
- class `SocketStream`
- class `stream_line_reader`

Перечисления

- enum class `EncodingType` { `None = 0` , `Gzip` , `Brotli` , `Zstd` }

Функции

- template<class T, class... Args>
std::enable_if<!std::is_array< T >::value, std::unique_ptr< T > >::type `make_unique` (Args &&...args)
- template<class T>
std::enable_if< std::is_array< T >::value, std::unique_ptr< T > >::type `make_unique` (std::size_t n)
- bool `set_socket_opt_impl` (socket_t sock, int level, int optname, const void *optval, socklen_t optlen)
- bool `set_socket_opt` (socket_t sock, int level, int optname, int opt)
- bool `set_socket_opt_time` (socket_t sock, int level, int optname, time_t sec, time_t usec)
- ssize_t `write_headers` (Stream &strm, const Headers &headers)
- template<typename T, typename U>
void `duration_to_sec_and_usec` (const T &duration, U callback)
- template<size_t N>
constexpr size_t `str_len` (const char(&)[N])
- bool `is_numeric` (const std::string &str)
- uint64_t `get_header_value_u64` (const Headers &headers, const std::string &key, uint64_t def, size_t id, bool &is_invalid_value)
- uint64_t `get_header_value_u64` (const Headers &headers, const std::string &key, uint64_t def, size_t id)
- std::string `encode_query_param` (const std::string &value)
- std::string `decode_url` (const std::string &s, bool convert_plus_to_space)
- std::string `trim_copy` (const std::string &s)
- void `divide` (const char *data, std::size_t size, char d, std::function< void(const char *, std::size_t, const char *, std::size_t) > fn)
- void `divide` (const std::string &str, char d, std::function< void(const char *, std::size_t, const char *, std::size_t) > fn)
- void `split` (const char *b, const char *e, char d, std::function< void(const char *, const char *) > fn)
- void `split` (const char *b, const char *e, char d, std::size_t m, std::function< void(const char *, const char *) > fn)
- bool `process_client_socket` (socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, time_t max_timeout_msec, std::chrono::time_point< std::chrono::steady_clock > start_time, std::function< bool(Stream &) > callback)
- socket_t `create_client_socket` (const std::string &host, const std::string &ip, int port, int address_family, bool tcp_nodelay, bool ipv6_v6only, SocketOptions socket_options, time_t connection_timeout_sec, time_t connection_timeout_usec, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, const std::string &intf, Error &error)
- const char * `get_header_value` (const Headers &headers, const std::string &key, const char *def, size_t id)
- std::string `params_to_query_str` (const Params ¶ms)
- void `parse_query_text` (const char *data, std::size_t size, Params ¶ms)
- void `parse_query_text` (const std::string &s, Params ¶ms)
- bool `parse_multipart_boundary` (const std::string &content_type, std::string &boundary)
- bool `parse_range_header` (const std::string &s, Ranges &ranges)
- int `close_socket` (socket_t sock)
- ssize_t `send_socket` (socket_t sock, const void *ptr, size_t size, int flags)
- ssize_t `read_socket` (socket_t sock, void *ptr, size_t size, int flags)
- EncodingType `encoding_type` (const Request &req, const Response &res)
- bool `is_hex` (char c, int &v)
- bool `from_hex_to_i` (const std::string &s, size_t i, size_t cnt, int &val)
- std::string `from_i_to_hex` (size_t n)
- size_t `to_utf8` (int code, char *buff)

- `std::string base64_encode (const std::string &in)`
- `bool is_valid_path (const std::string &path)`
- `std::string encode_url (const std::string &s)`
- `std::string file_extension (const std::string &path)`
- `bool is_space_or_tab (char c)`
- `std::pair< size_t, size_t > trim (const char *b, const char *e, size_t left, size_t right)`
- `std::string trim_double_quotes_copy (const std::string &s)`
- template<typename T>
 `ssize_t handle_EINTR (T fn)`
- `int poll_wrapper (struct pollfd *fds, nfds_t nfds, int timeout)`
- template<bool Read>
 `ssize_t select_impl (socket_t sock, time_t sec, time_t usec)`
- `ssize_t select_read (socket_t sock, time_t sec, time_t usec)`
- `ssize_t select_write (socket_t sock, time_t sec, time_t usec)`
- `Error wait_until_socket_is_ready (socket_t sock, time_t sec, time_t usec)`
- `bool is_socket_alive (socket_t sock)`
- `bool keep_alive (const std::atomic< socket_t > &svr_sock, socket_t sock, time_t keep_alive_ ← timeout_sec)`
- template<typename T>
 `bool process_server_socket_core (const std::atomic< socket_t > &svr_sock, socket_t sock, size_ ← _t keep_alive_max_count, time_t keep_alive_timeout_sec, T callback)`
- template<typename T>
 `bool process_server_socket (const std::atomic< socket_t > &svr_sock, socket_t sock, size_ ← t keep_alive_max_count, time_t keep_alive_timeout_sec, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, T callback)`
- `int shutdown_socket (socket_t sock)`
- `std::string escape_abstract_namespace_unix_domain (const std::string &s)`
- `std::string unescape_abstract_namespace_unix_domain (const std::string &s)`
- template<typename BindOrConnect>
 `socket_t create_socket (const std::string &host, const std::string &ip, int port, int address_family, int socket_flags, bool tcp_nodelay, bool ipv6_v6only, SocketOptions socket_options, BindOr← Connect bind_or_connect)`
- `void set_nonblocking (socket_t sock, bool nonblocking)`
- `bool is_connection_error ()`
- `bool bind_ip_address (socket_t sock, const std::string &host)`
- `std::string if2ip (int address_family, const std::string &ifn)`
- `bool get_ip_and_port (const struct sockaddr_storage &addr, socklen_t addr_len, std::string &ip, int &port)`
- `void get_local_ip_and_port (socket_t sock, std::string &ip, int &port)`
- `void get_remote_ip_and_port (socket_t sock, std::string &ip, int &port)`
- `constexpr unsigned int str2tag_core (const char *s, size_t l, unsigned int h)`
- `unsigned int str2tag (const std::string &s)`
- `std::string find_content_type (const std::string &path, const std::map< std::string, std::string > &user_data, const std::string &default_content_type)`
- `bool can_compress_content_type (const std::string &content_type)`
- `bool has_header (const Headers &headers, const std::string &key)`
- template<typename T>
 `bool parse_header (const char *beg, const char *end, T fn)`
- `bool read_headers (Stream &strm, Headers &headers)`
- `bool read_content_with_length (Stream &strm, uint64_t len, Progress progress, ContentReceiverWithProgress out)`
- `void skip_content_with_length (Stream &strm, uint64_t len)`
- `bool read_content_without_length (Stream &strm, ContentReceiverWithProgress out)`
- template<typename T>
 `bool read_content_chunked (Stream &strm, T &x, ContentReceiverWithProgress out)`
- `bool is_chunked_transfer_encoding (const Headers &headers)`

- template<typename T, typename U>
bool `prepare_content_receiver` (T &x, int &status, `ContentReceiverWithProgress` receiver, bool decompress, U callback)
- template<typename T>
bool `read_content` (Stream &strm, T &x, size_t payload_max_length, int &status, `Progress` progress, `ContentReceiverWithProgress` receiver, bool decompress)
- ssize_t `write_request_line` (Stream &strm, const std::string &method, const std::string &path)
- ssize_t `write_response_line` (Stream &strm, int status)
- bool `write_data` (Stream &strm, const char *d, size_t l)
- template<typename T>
bool `write_content` (Stream &strm, const `ContentProvider` &content_provider, size_t offset, size_t length, T is_shutting_down, `Error` &error)
- template<typename T>
bool `write_content` (Stream &strm, const `ContentProvider` &content_provider, size_t offset, size_t length, const T &is_shutting_down)
- template<typename T>
bool `write_content_without_length` (Stream &strm, const `ContentProvider` &content_provider, const T &is_shutting_down)
- template<typename T, typename U>
bool `write_content_chunked` (Stream &strm, const `ContentProvider` &content_provider, const T &is_shutting_down, U &compressor, `Error` &error)
- template<typename T, typename U>
bool `write_content_chunked` (Stream &strm, const `ContentProvider` &content_provider, const T &is_shutting_down, U &compressor)
- template<typename T>
bool `redirect` (T &cli, `Request` &req, `Response` &res, const std::string &path, const std::string &location, `Error` &error)
- void `parse_disposition_params` (const std::string &s, `Params` ¶ms)
- std::string `random_string` (size_t length)
- std::string `make_multipart_data_boundary` ()
- bool `is_multipart_boundary_chars_valid` (const std::string &boundary)
- template<typename T>
std::string `serialize_multipart_formdata_item_begin` (const T &item, const std::string &boundary)
- std::string `serialize_multipart_formdata_item_end` ()
- std::string `serialize_multipart_formdata_finish` (const std::string &boundary)
- std::string `serialize_multipart_formdata_get_content_type` (const std::string &boundary)
- std::string `serialize_multipart_formdata` (const `MultipartFormDataItems` &items, const std::string &boundary, bool finish=true)
- bool `range_error` (`Request` &req, `Response` &res)
- std::pair<size_t, size_t> `get_range_offset_and_length` (Range r, size_t content_length)
- std::string `make_content_range_header_field` (const std::pair<size_t, size_t> &offset_and_length, size_t content_length)
- template<typename SToken, typename CToken, typename Content>
bool `process_multipart_ranges_data` (const `Request` &req, const std::string &boundary, const std::string &content_type, size_t content_length, SToken stoken, CToken ctoken, Content content)
- void `make_multipart_ranges_data` (const `Request` &req, `Response` &res, const std::string &boundary, const std::string &content_type, size_t content_length, std::string &data)
- size_t `get_multipart_ranges_data_length` (const `Request` &req, const std::string &boundary, const std::string &content_type, size_t content_length)
- template<typename T>
bool `write_multipart_ranges_data` (Stream &strm, const `Request` &req, `Response` &res, const std::string &boundary, const std::string &content_type, size_t content_length, const T &is_shutting_down)
- bool `expect_content` (const `Request` &req)
- bool `has_crlf` (const std::string &s)

- bool `parse_www_authenticate` (const `Response` &res, std::map< std::string, std::string > &auth, bool is_proxy)
- void `calc_actual_timeout` (time_t max_timeout_msec, time_t duration_msec, time_t timeout_sec, time_t timeout_usec, time_t &actual_timeout_sec, time_t &actual_timeout_usec)

5.3.1 Перечисления

5.3.1.1 EncodingType

enum class `httplib::detail::EncodingType` [strong]

Элементы перечислений

None	
Gzip	
Brotli	
Zstd	

См. определение в файле `httplib.h` строка 2455

02455 { `None` = 0, `Gzip`, `Brotli`, `Zstd` };

5.3.2 Функции

5.3.2.1 base64_encode()

```
std::string httplib::detail::base64_encode (
    const std::string & in) [inline]
```

См. определение в файле `httplib.h` строка 2780

```
02780     static const auto lookup =
02781         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
02782
02783     std::string out;
02784     out.reserve(in.size());
02785
02786     auto val = 0;
02787     auto valb = -6;
02788
02789     for (auto c : in) {
02790         val = (val << 8) + static_cast<uint8_t>(c);
02791         valb += 8;
02792         while (valb >= 0) {
02793             out.push_back(lookup[(val >> valb) & 0x3F]);
02794             valb -= 6;
02795         }
02796     }
02797
02798     if (valb > -6) { out.push_back(lookup[((val << 8) >> (valb + 8)) & 0x3F]); }
02799
02800     while (out.size() % 4) {
02801         out.push_back('=');
02802     }
02803
02804     return out;
02805 }
```

Граф вызова функции:



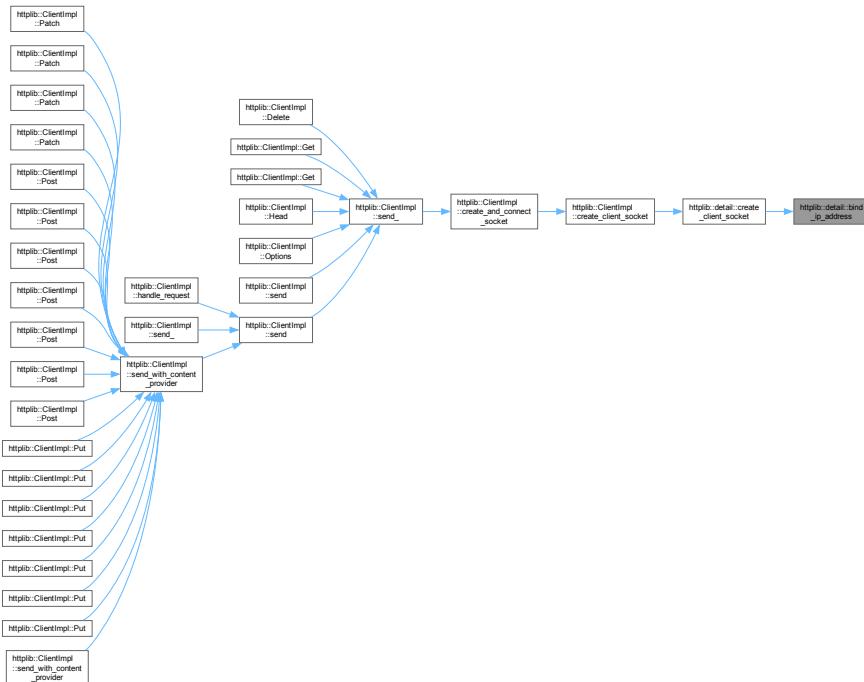
5.3.2.2 `bind_ip_address()`

```
bool httpplib::detail::bind_ip_address (
    socket_t sock,
    const std::string & host) [inline]
```

См. определение в файле `httpplib.h` строка 3680

```
03680                                     {
03681     struct addrinfo hints;
03682     struct addrinfo *result;
03683
03684     memset(&hints, 0, sizeof(struct addrinfo));
03685     hints.ai_family = AF_UNSPEC;
03686     hints.ai_socktype = SOCK_STREAM;
03687     hints.ai_protocol = 0;
03688
03689     if (getaddrinfo(host.c_str(), "0", &hints, &result)) { return false; }
03690     auto se = detail::scope_exit([&] { freeaddrinfo(result); });
03691
03692     auto ret = false;
03693     for (auto rp = result; rp; rp = rp->ai_next) {
03694         const auto &zai = *rp;
03695         if (!::bind(sock, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen))) {
03696             ret = true;
03697             break;
03698         }
03699     }
03700
03701     return ret;
03702 }
```

Граф вызова функции:



5.3.2.3 `calc_actual_timeout()`

```
void httpplib::detail::calc_actual_timeout (
    time_t max_timeout_msec,
```

```
time_t duration_msec,
time_t timeout_sec,
time_t timeout_usec,
time_t & actual_timeout_sec,
time_t & actual_timeout_usec) [inline]
```

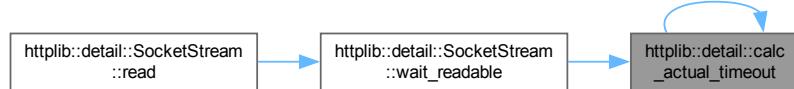
См. определение в файле [httpplib.h](#) строка 6049

```
06052     {
06053     auto timeout_msec = (timeout_sec * 1000) + (timeout_usec / 1000);
06054
06055     auto actual_timeout_msec =
06056         (std::min)(max_timeout_msec - duration_msec, timeout_msec);
06057
06058     if (actual_timeout_msec < 0) { actual_timeout_msec = 0; }
06059
06060     actual_timeout_sec = actual_timeout_msec / 1000;
06061     actual_timeout_usec = (actual_timeout_msec % 1000) * 1000;
06062 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.4 `can_compress_content_type()`

```
bool httpplib::detail::can_compress_content_type (
    const std::string & content_type) [inline]
```

См. определение в файле [httpplib.h](#) строка 3954

```
03954     {
03955     using udl::operator""_t;
03956
03957     auto tag = str2tag(content_type);
03958
03959     switch (tag) {
03960     case "image/svg+xml"_t:
03961     case "application/javascript"_t:
03962     case "application/json"_t:
03963     case "application/xml"_t:
```

```

03964 case "application/protobuf" _t:
03965 case "application/xhtml+xml" _t: return true;
03966
03967 case "text/event-stream" _t: return false;
03968
03969 default: return !content_type.rfind("text/", 0);
03970 }
03971 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.5 `close_socket()`

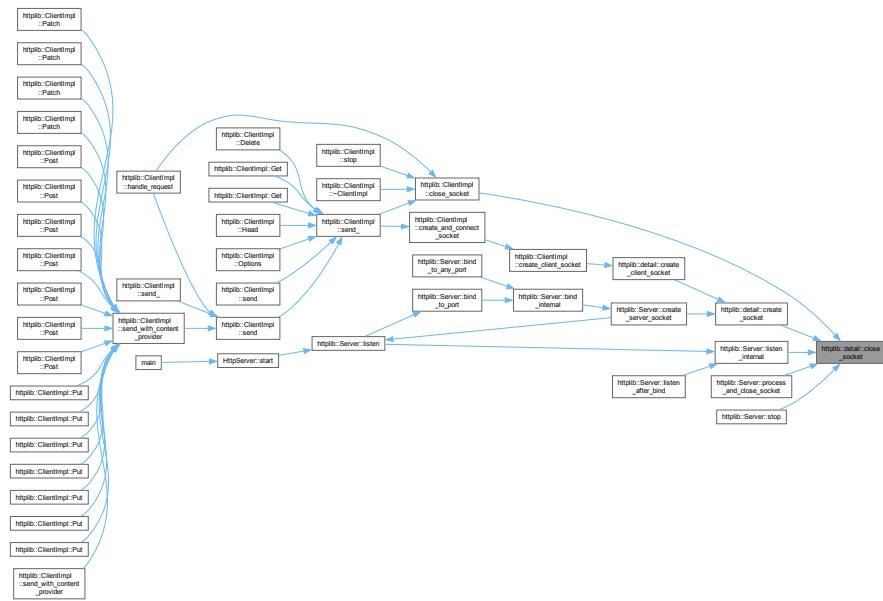
```
int httpplib::detail::close_socket (
    socket_t sock) [inline]
```

См. определение в файле `httpplib.h` строка 3236

```

03236 {
03237 #ifdef _WIN32
03238     return closesocket(sock);
03239 #else
03240     return close(sock);
03241 #endif
03242 }
```

Граф вызова функции:



5.3.2.6 create_client_socket()

```
socket_t httpclient::detail::create_client_socket (
    const std::string & host,
    const std::string & ip,
    int port,
    int address_family,
    bool tcp_nodelay,
    bool ipv6_v6only,
    SocketOptions socket_options,
    time_t connection_timeout_sec,
    time_t connection_timeout_usec,
    time_t read_timeout_sec,
    time_t read_timeout_usec,
    time_t write_timeout_sec,
    time_t write_timeout_usec,
    const std::string & intf,
    Error & error) [inline]
```

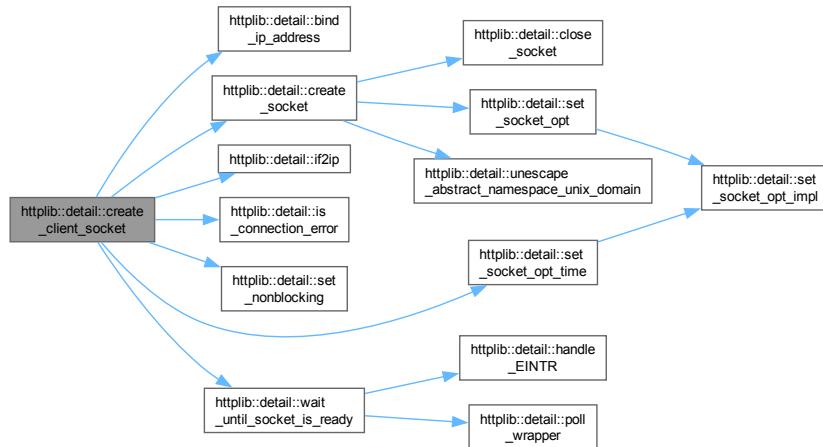
См. определение в файле `httpplib.h` строка 3746

```
03752 {  
03753     auto sock = create_socket(  
03754         host, ip, port, address_family, 0, tcp_nodelay, ipv6_v6only,  
03755         std::move(socket_options),  
03756         [&](socket_t sock2, struct sockaddr &ai, bool &qexit) -> bool {  
03757             if (!intf.empty()) {  
03758 #ifdef USE_IF2IP  
03759                 auto ip_from_if = if2ip(address_family, intf);  
03760                 if (ip_from_if.empty()) { ip_from_if = intf; }  
03761                 if (!bind_ip_address(sock2, ip_from_if)) {  
03762                     error = Error::BindIPAddress;  
03763                     return false;  
03764                 }  
03765 #endif  
03766             }  
03767         }  
03768         set_nonblocking(sock2, true);  
03769     }  
03770 }
```

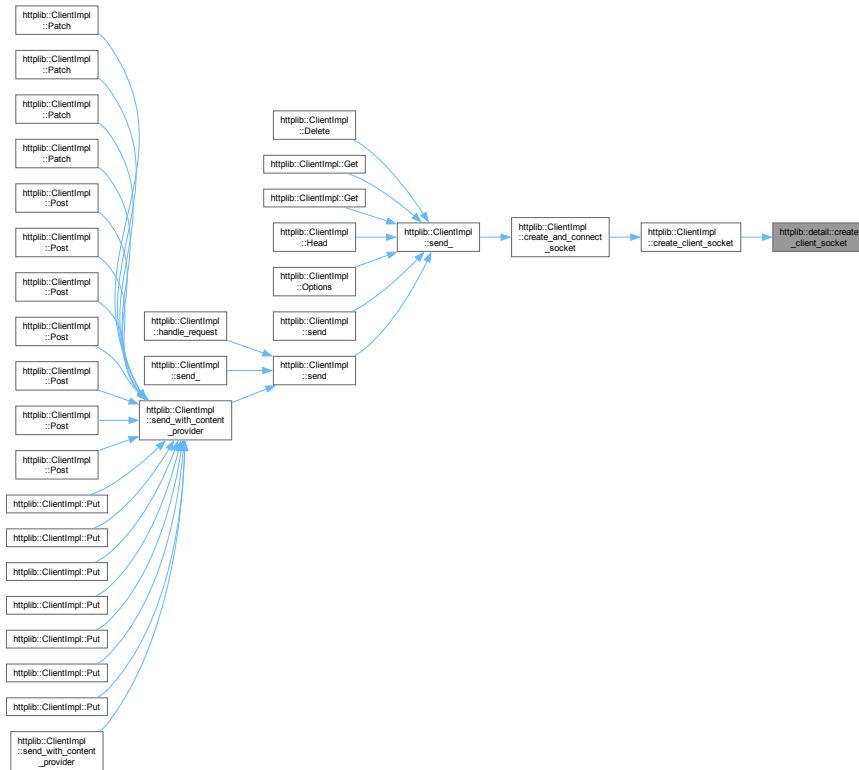
```

03769
03770     auto ret =
03771         ::connect(sock2, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen));
03772
03773     if (ret < 0) {
03774         if (is_connection_error()) {
03775             error = Error::Connection;
03776             return false;
03777         }
03778         error = wait_until_socket_is_ready(sock2, connection_timeout_sec,
03779                                         connection_timeout_usec);
03780         if (error != Error::Success) {
03781             if (error == Error::ConnectionTimeout) { quit = true; }
03782             return false;
03783         }
03784     }
03785
03786     set_nonblocking(sock2, false);
03787     set_socket_opt_time(sock2, SOL_SOCKET, SO_RCVTIMEO, read_timeout_sec,
03788                          read_timeout_usec);
03789     set_socket_opt_time(sock2, SOL_SOCKET, SO_SNDTIMEO, write_timeout_sec,
03790                          write_timeout_usec);
03791
03792     error = Error::Success;
03793     return true;
03794 });
03795
03796 if (sock != INVALID_SOCKET) {
03797     error = Error::Success;
03798 } else {
03799     if (error == Error::Success) { error = Error::Connection; }
03800 }
03801
03802 return sock;
03803 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.7 `create_socket()`

```

template<typename BindOrConnect>
socket_t httpplib::detail::create_socket (
    const std::string & host,
    const std::string & ip,
    int port,
    int address_family,
    int socket_flags,
    bool tcp_nodelay,
    bool ipv6_v6only,
    SocketOptions socket_options,
    BindOrConnect bind_or_connect)

```

См. определение в файле `httpplib.h` строка 3520

```

03523
03524 // Get address info
03525 const char *node = nullptr;
03526 struct addrinfo hints;
03527 struct addrinfo *result;
03528
03529 memset(&hints, 0, sizeof(struct addrinfo));
03530 hints.ai_socktype = SOCK_STREAM;
03531 hints.ai_protocol = IPPROTO_IP;
03532
03533 if (!ip.empty()) {
03534     node = ip.c_str();
03535     // Ask getaddrinfo to convert IP in c-string to address
03536     hints.ai_family = AF_UNSPEC;
03537     hints.ai_flags = AI_NUMERICHOST;

```

```

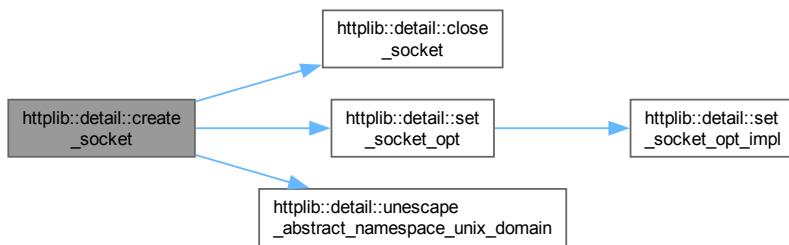
03538 } else {
03539     if (!host.empty()) { node = host.c_str(); }
03540     hints.ai_family = address_family;
03541     hints.ai_flags = socket_flags;
03542 }
03543
03544 if (hints.ai_family == AF_UNIX) {
03545     const auto addrlen = host.length();
03546     if (addrlen > sizeof(sockaddr_un::sun_path)) { return INVALID_SOCKET; }
03547
03548 #ifdef SOCK_CLOEXEC
03549     auto sock = socket(hints.ai_family, hints.ai_socktype | SOCK_CLOEXEC,
03550                         hints.ai_protocol);
03551 #else
03552     auto sock = socket(hints.ai_family, hints.ai_socktype, hints.ai_protocol);
03553 #endif
03554
03555     if (sock != INVALID_SOCKET) {
03556         sockaddr_un addr{{}};
03557         addr.sun_family = AF_UNIX;
03558
03559         auto unescaped_host = unescape_abstract_namespace_unix_domain(host);
03560         std::copy(unescaped_host.begin(), unescaped_host.end(), addr.sun_path);
03561
03562         hints.ai_addr = reinterpret_cast<sockaddr*>(&addr);
03563         hints.ai_addrlen = static_cast<socklen_t>(
03564             sizeof(addr) - sizeof(addr.sun_path) + addrlen);
03565
03566 #ifndef SOCK_CLOEXEC
03567 #ifndef _WIN32
03568     fcntl(sock, F_SETFD, FD_CLOEXEC);
03569 #endif
03570 #endif
03571
03572     if (socket_options) { socket_options(sock); }
03573
03574 #ifdef _WIN32
03575     // Setting SO_REUSEADDR seems not to work well with AF_UNIX on windows, so
03576     // remove the option.
03577     detail::set_socket_opt(sock, SOL_SOCKET, SO_REUSEADDR, 0);
03578 #endif
03579
03580     bool dummy;
03581     if (!bind_or_connect(sock, hints, dummy)) {
03582         close_socket(sock);
03583         sock = INVALID_SOCKET;
03584     }
03585 }
03586     return sock;
03587 }
03588
03589 auto service = std::to_string(port);
03590
03591     if (getaddrinfo(node, service.c_str(), &hints, &result)) {
03592 #if defined __linux__ && !defined __ANDROID__
03593     res_init();
03594 #endif
03595     return INVALID_SOCKET;
03596 }
03597     auto se = detail::scope_exit([&] { freeaddrinfo(result); });
03598
03599     for (auto rp = result; rp; rp = rp->ai_next) {
03600         // Create a socket
03601 #ifdef _WIN32
03602     auto sock =
03603         WSASocketW(rp->ai_family, rp->ai_socktype, rp->ai_protocol, nullptr, 0,
03604                     WSA_FLAG_NO_HANDLE_INHERIT | WSA_FLAG_OVERLAPPED);
03619     if (sock == INVALID_SOCKET) {
03620         sock = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
03621     }
03622 #else
03623
03624 #ifdef SOCK_CLOEXEC
03625     auto sock =
03626         socket(rp->ai_family, rp->ai_socktype | SOCK_CLOEXEC, rp->ai_protocol);
03627 #else
03628     auto sock = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
03629 #endif
03630
03631 #endif
03632     if (sock == INVALID_SOCKET) { continue; }
03633
03634 #if !defined _WIN32 && !defined SOCK_CLOEXEC
03635     if (fcntl(sock, F_SETFD, FD_CLOEXEC) == -1) {
03636         close_socket(sock);
03637         continue;
03638     }

```

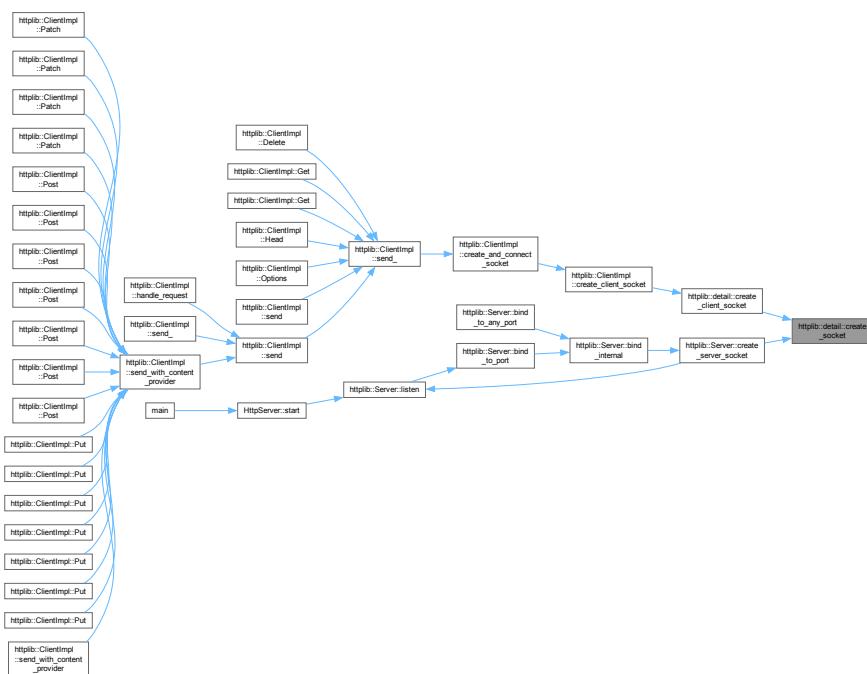
```

03639 #endif
03640
03641     if (tcp_nodelay) { set_socket_opt(sock, IPPROTO_TCP, TCP_NODELAY, 1); }
03642
03643     if (rp->ai_family == AF_INET6)
03644         set_socket_opt(sock, IPPROTO_IPV6, IPV6_V6ONLY, ipv6_v6only ? 1 : 0);
03645     }
03646
03647     if (socket_options) { socket_options(sock); }
03648
03649 // bind or connect
03650 auto quit = false;
03651 if (bind_or_connect(sock, *rp, quit)) { return sock; }
03652
03653 close_socket(sock);
03654
03655 if (quit) { break; }
03656 }
03657
03658 return INVALID_SOCKET;
03659 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.8 `decode_url()`

```
std::string httpplib::detail::decode_url (
    const std::string & s,
    bool convert_plus_to_space) [inline]
```

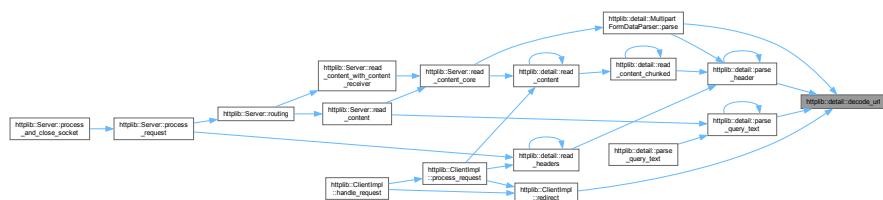
См. определение в файле [httpplib.h](#) строка 2918

```
02919
02920     std::string result;
02921
02922     for (size_t i = 0; i < s.size(); i++) {
02923         if (s[i] === '%' && i + 1 < s.size()) {
02924             if (s[i + 1] === 'u') {
02925                 auto val = 0;
02926                 if (from_hex_to_i(s, i + 2, 4, val)) {
02927                     // 4 digits Unicode codes
02928                     char buff[4];
02929                     size_t len = to_utf8(val, buff);
02930                     if (len > 0) { result.append(buff, len); }
02931                     i += 5; // 'u0000'
02932                 } else {
02933                     result += s[i];
02934                 }
02935             } else {
02936                 auto val = 0;
02937                 if (from_hex_to_i(s, i + 1, 2, val)) {
02938                     result += static_cast<char>(val);
02939                     i += 2; // '00'
02940                 } else {
02941                     result += s[i];
02942                 }
02943             }
02944         } else if (convert_plus_to_space && s[i] === '+') {
02945             result += ' ';
02946         } else {
02947             result += s[i];
02948         }
02949     }
02950
02951     return result;
02952 }
```

Граф вызовов:



Граф вызова функции:



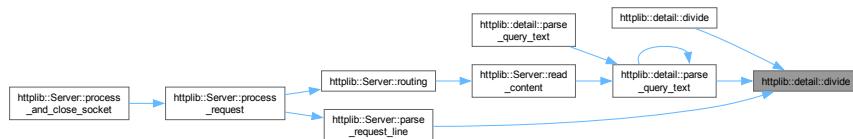
5.3.2.9 `divide()` [1/2]

```
void httpplib::detail::divide (
    const char * data,
    std::size_t size,
    char d,
    std::function< void(const char *, std::size_t, const char *, std::size_t)> fn) [inline]
```

См. определение в файле `httpplib.h` строка 2988

```
02990     {
02991     const auto it = std::find(data, data + size, d);
02992     const auto found = static_cast<std::size_t>(it != data + size);
02993     const auto lhs_data = data;
02994     const auto lhs_size = static_cast<std::size_t>(it - data);
02995     const auto rhs_data = it + found;
02996     const auto rhs_size = size - lhs_size - found;
02997
02998     fn(lhs_data, lhs_size, rhs_data, rhs_size);
02999 }
```

Граф вызова функции:

5.3.2.10 `divide()` [2/2]

```
void httpplib::detail::divide (
    const std::string & str,
    char d,
    std::function< void(const char *, std::size_t, const char *, std::size_t)> fn) [inline]
```

См. определение в файле `httpplib.h` строка 3002

```
03004     {
03005     divide(str.data(), str.size(), d, std::move(fn));
03006 }
```

Граф вызовов:



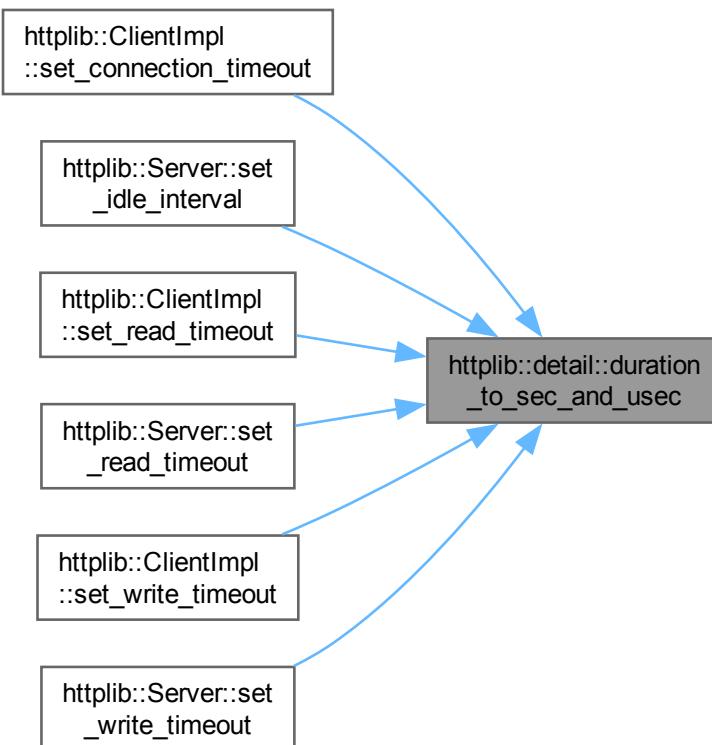
5.3.2.11 duration_to_sec_and_usec()

```
template<typename T, typename U>
void httpplib::detail::duration_to_sec_and_usec (
    const T & duration,
    U callback) [inline]
```

См. определение в файле [httpplib.h](#) строка 2052

```
02052     {
02053     auto sec = std::chrono::duration_cast<std::chrono::seconds>(duration).count();
02054     auto usec = std::chrono::duration_cast<std::chrono::microseconds>(
02055         duration - std::chrono::seconds(sec))
02056         .count();
02057     callback(static_cast<time_t>(sec), static_cast<time_t>(usec));
02058 }
```

Граф вызова функции:



5.3.2.12 encode_query_param()

```
std::string httpplib::detail::encode_query_param (
    const std::string & value) [inline]
```

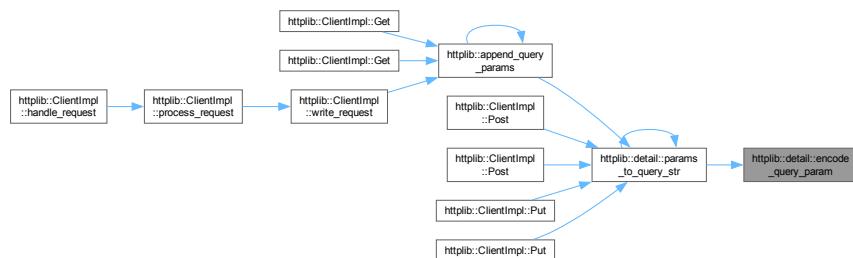
См. определение в файле [httpplib.h](#) строка 2865

```
02865     {
02866     std::ostringstream escaped;
```

```

02867     escaped.fill('0');
02868     escaped < std::hex;
02869
02870     for (auto c : value) {
02871         if (std::isalnum(static_cast<uint8_t>(c)) || c == '-' || c == '_' ||
02872             c == '.' || c == '!' || c == '~' || c == '*' || c == '\'') ||
02873             c == ')') {
02874             escaped << c;
02875         } else {
02876             escaped << std::uppercase;
02877             escaped << '%' << std::setw(2)
02878                 << static_cast<int>(static_cast<unsigned char>(c));
02879             escaped << std::nouppercase;
02880         }
02881     }
02882
02883     return escaped.str();
02884 }
```

Граф вызова функции:



5.3.2.13 encode_url()

```
std::string httplib::detail::encode_url (
    const std::string & s) [inline]
```

См. определение в файле [httplib.h](#) строка 2886

```

02886     std::string result;
02887     result.reserve(s.size());
02888
02889     for (size_t i = 0; s[i]; i++) {
02890         switch (s[i]) {
02891             case '%': result += "%20"; break;
02892             case '+': result += "%2B"; break;
02893             case '\r': result += "%0D"; break;
02894             case '\n': result += "%0A"; break;
02895             case '\\': result += "%27"; break;
02896             case ';': result += "%2C"; break;
02897             // case '?': result += "%3A"; break; // ok? probably...
02898             case ':': result += "%3B"; break;
02899             default:
02900                 auto c = static_cast<uint8_t>(s[i]);
02901                 if (c >= 0x80) {
02902                     result += '%';
02903                     char hex[4];
02904                     auto len = sprintf(hex, sizeof(hex) - 1, "%02X", c);
02905                     assert(len == 2);
02906                     result.append(hex, static_cast<size_t>(len));
02907                 } else {
02908                     result += s[i];
02909                 }
02910             }
02911         break;
02912     }
02913 }
02914
02915 return result;
02916 }
```

Граф вызова функции:



5.3.2.14 `encoding_type()`

```
EncodingType httpplib::detail::encoding_type (
    const Request & req,
    const Response & res) [inline]
```

См. определение в файле [httpplib.h](#) строка 3973

```

03973     auto ret = {  

03974         detail::can_compress_content_type(res.get_header_value("Content-Type"));  

03975         if (!ret) { return EncodingType::None; }  

03977  

03978         const auto &s = req.get_header_value("Accept-Encoding");  

03979         (void)(s);  

03980  

03981 #ifdef CPPHTTPPLIB_BROTLI_SUPPORT  

03982 // TODO: 'Accept-Encoding' has br, not br;q=0  

03983         ret = s.find("br") != std::string::npos;  

03984         if (ret) { return EncodingType::Brotli; }  

03985     #endif  

03986  

03987 #ifdef CPPHTTPPLIB_ZLIB_SUPPORT  

03988 // TODO: 'Accept-Encoding' has gzip, not gzip;q=0  

03989         ret = s.find("gzip") != std::string::npos;  

03990         if (ret) { return EncodingType::Gzip; }  

03991     #endif  

03992  

03993 #ifdef CPPHTTPPLIB_ZSTD_SUPPORT  

03994 // TODO: 'Accept-Encoding' has zstd, not zstd;q=0  

03995         ret = s.find("zstd") != std::string::npos;  

03996         if (ret) { return EncodingType::Zstd; }  

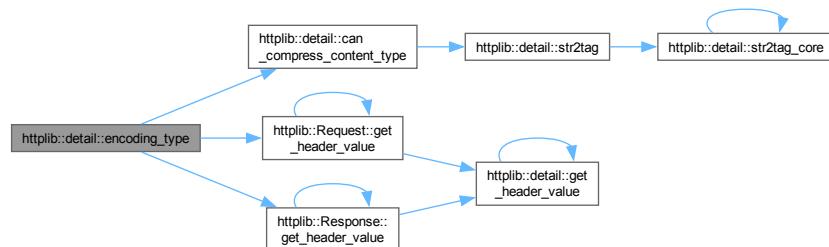
03997     #endif  

03998  

03999     return EncodingType::None;  

04000 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.15 `escape_abstract_namespace_unix_domain()`

```
std::string httpplib::detail::escape_abstract_namespace_unix_domain (
    const std::string & s) [inline]
```

См. определение в файле `httpplib.h` строка 3500

```
03500 if (s.size() > 1 && s[0] == '\0') {
03501     auto ret = s;
03502     ret[0] = '@';
03503     return ret;
03504 }
03505 return s;
03506 }
```

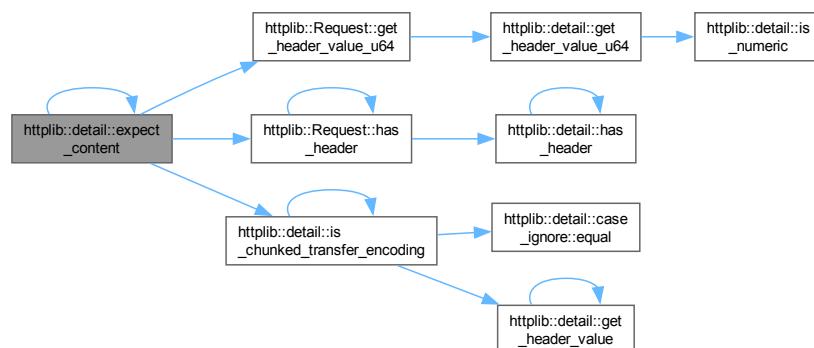
5.3.2.16 `expect_content()`

```
bool httpplib::detail::expect_content (
    const Request & req) [inline]
```

См. определение в файле `httpplib.h` строка 5468

```
05468 {
05469     if (req.method == "POST" || req.method == "PUT" || req.method == "PATCH" ||
05470         req.method == "DELETE") {
05471         return true;
05472     }
05473     if (req.has_header("Content-Length") &&
05474         req.get_header_value_u64("Content-Length") > 0) {
05475         return true;
05476     }
05477     if (is_chunked_transfer_encoding(req.headers)) { return true; }
05478     return false;
05479 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.17 file_extension()

```
std::string httpplib::detail::file_extension (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 2955

```
02955                                     {
02956     std::smatch m;
02957     thread_local auto re = std::regex("^.([a-zA-Z0-9]+)$");
02958     if (std::regex_search(path, m, re)) { return m[1].str(); }
02959     return std::string();
02960 }
```

Граф вызова функции:



5.3.2.18 find_content_type()

```
std::string httpplib::detail::find_content_type (
    const std::string & path,
    const std::map< std::string, std::string > & user_data,
    const std::string & default_content_type) [inline]
```

См. определение в файле `httpplib.h` строка 3889

```
03891                                     {
03892     auto ext = file_extension(path);
03893
03894     auto it = user_data.find(ext);
03895     if (it != user_data.end()) { return it->second; }
03896
03897     using udl::operator""_t;
03898
03899     switch (str2tag(ext)) {
03900     default: return default_content_type;
03901
03902     case "css"_t: return "text/css";
03903     case "csv"_t: return "text/csv";
03904     case "htm"_t:
03905     case "html"_t: return "text/html";
03906     case "js"_t:
03907     case "mjs"_t: return "text/javascript";
03908     case "txt"_t: return "text/plain";
03909     case "vtt"_t: return "text/vtt";
03910
03911     case "apng"_t: return "image/apng";
03912     case "avif"_t: return "image/avif";
03913     case "bmp"_t: return "image/bmp";
```

```

03914 case "gif" _t: return "image/gif";
03915 case "png" _t: return "image/png";
03916 case "svg" _t: return "image/svg+xml";
03917 case "webp" _t: return "image/webp";
03918 case "ico" _t: return "image/x-icon";
03919 case "tif" _t: return "image/tiff";
03920 case "tiff" _t: return "image/tiff";
03921 case "jpg" _t:
03922 case "jpeg" _t: return "image/jpeg";
03923
03924 case "mp4" _t: return "video/mp4";
03925 case "mpeg" _t: return "video/mpeg";
03926 case "webm" _t: return "video/webm";
03927
03928 case "mp3" _t: return "audio/mp3";
03929 case "mpga" _t: return "audio/mpeg";
03930 case "weba" _t: return "audio/webm";
03931 case "wav" _t: return "audio/wave";
03932
03933 case "otf" _t: return "font/otf";
03934 case "ttf" _t: return "font/ttf";
03935 case "woff" _t: return "font/woff";
03936 case "woff2" _t: return "font/woff2";
03937
03938 case "7z" _t: return "application/x-7z-compressed";
03939 case "atom" _t: return "application/atom+xml";
03940 case "pdf" _t: return "application/pdf";
03941 case "json" _t: return "application/json";
03942 case "rss" _t: return "application/rss+xml";
03943 case "tar" _t: return "application/x-tar";
03944 case "xht" _t:
03945 case "xhtml" _t: return "application/xhtml+xml";
03946 case "xslt" _t: return "application/xslt+xml";
03947 case "xml" _t: return "application/xml";
03948 case "gz" _t: return "application/gzip";
03949 case "zip" _t: return "application/zip";
03950 case "wasm" _t: return "application/wasm";
03951 }
03952 }

```

Граф вызовов:



Граф вызова функции:



5.3.2.19 `from_hex_to_i()`

```

bool http://detail::from_hex_to_i (
    const std::string & s,
    size_t i,

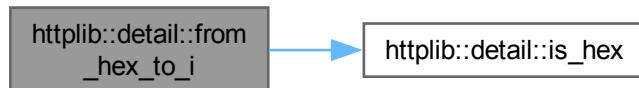
```

```
size_t cnt,
int & val) [inline]
```

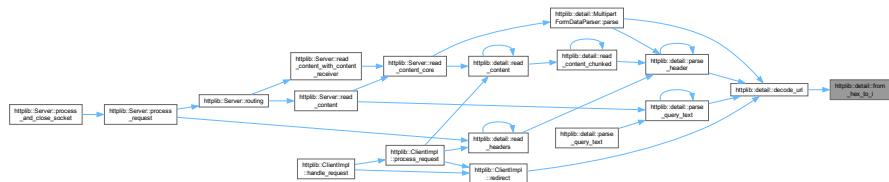
См. определение в файле [httpplib.h](#) строка 2719

```
02720     {
02721     if (i >= s.size()) { return false; }
02722
02723     val = 0;
02724     for (; cnt; i++, cnt--) {
02725         if (!s[i]) { return false; }
02726         auto v = 0;
02727         if (is_hex(s[i], v)) {
02728             val = val * 16 + v;
02729         } else {
02730             return false;
02731         }
02732     }
02733     return true;
02734 }
```

Граф вызовов:



Граф вызова функции:



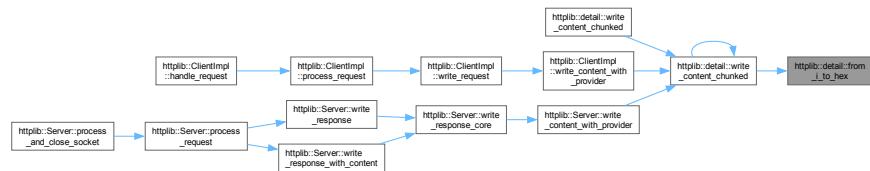
5.3.2.20 `from_i_to_hex()`

```
std::string httpplib::detail::from_i_to_hex (
    size_t n) [inline]
```

См. определение в файле [httpplib.h](#) строка 2736

```
02736     {
02737     static const auto charset = "0123456789abcdef";
02738     std::string ret;
02739     do {
02740         ret = charset[n & 15] + ret;
02741         n >= 4;
02742     } while (n > 0);
02743     return ret;
02744 }
```

Граф вызова функции:



5.3.2.21 `get_header_value()`

```
const char * httpplib::detail::get_header_value (
    const Headers & headers,
    const std::string & key,
    const char * def,
    size_t id) [inline]
```

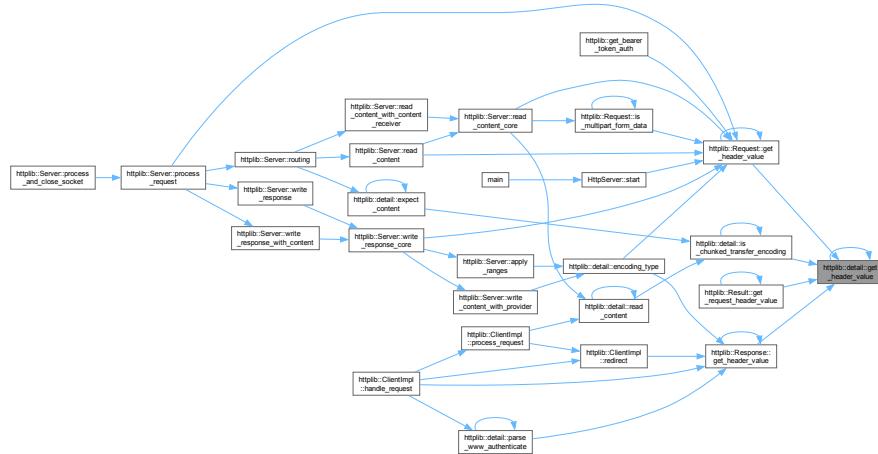
См. определение в файле [httpplib.h](#) строка 4266

```
04268     {
04269     auto rng = headers.equal_range(key);
04270     auto it = rng.first;
04271     std::advance(it, static_cast<ssize_t>(id));
04272     if (it != rng.second) { return it->second.c_str(); }
04273     return def;
04274 }
```

Граф вызовов:



Граф вызова функции:



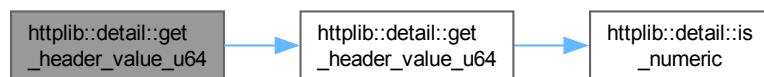
5.3.2.22 get_header_value_u64() [1/2]

```
uint64_t httplib::detail::get_header_value_u64 (
    const Headers & headers,
    const std::string & key,
    uint64_t def,
    size_t id) [inline]
```

См. определение в файле `httpplib.h` строка 2085

```
02087 {  
02088     bool dummy = false;  
02089     return get_header_value_u64(headers, key, def, id, dummy);  
02090 }
```

Граф вызовов:



5.3.2.23 get_header_value_u64() [2/2]

```
uint64_t httpplib::detail::get_header_value_u64 (
    const Headers & headers,
    const std::string & key,
    uint64_t def,
    size_t id,
    bool & is_invalid_value) [inline]
```

См. определение в файле `httpplib.h` строка 2068

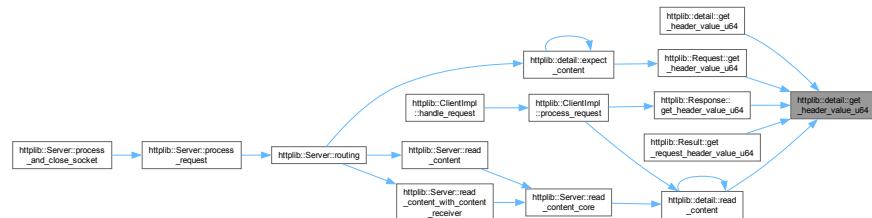
```

02070                                     {
02071     is_invalid_value = false;
02072     auto rng = headers.equal_range(key);
02073     auto it = rng.first;
02074     std::advance(it, static_cast<ssize_t>(id));
02075     if (it != rng.second) {
02076         if (is_numeric(it->second)) {
02077             return std::strtoull(it->second.data(), nullptr, 10);
02078         } else {
02079             is_invalid_value = true;
02080         }
02081     }
02082     return def;
02083 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.24 `get_ip_and_port()`

```

bool httpplib::detail::get_ip_and_port (
    const struct sockaddr_storage & addr,
    socklen_t addr_len,
    std::string & ip,
    int & port) [inline]
```

См. определение в файле `httpplib.h` строка 3805

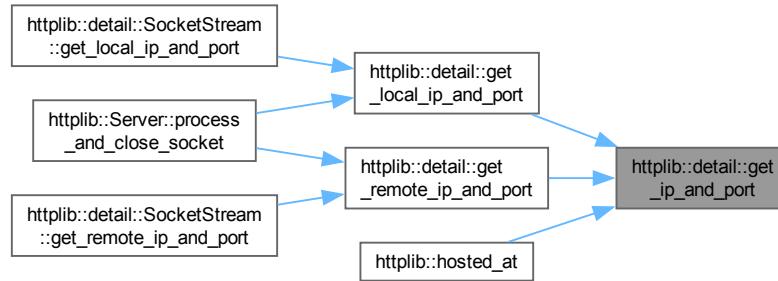
```

03806                                     {
03807     if (addr.ss_family == AF_INET) {
03808         port = ntohs(reinterpret_cast<const struct sockaddr_in*>(&addr)->sin_port);
03809     } else if (addr.ss_family == AF_INET6) {
03810         port =
03811         ntohs(reinterpret_cast<const struct sockaddr_in6*>(&addr)->sin6_port);
03812     } else {
03813         return false;
03814     }
03815
03816     std::array<char, NI_MAXHOST> ipstr{};
03817     if (getnameinfo(reinterpret_cast<const struct sockaddr*>(&addr), addr_len,
```

```

03818     ipstr.data(), static_cast<socklen_t>(ipstr.size()), nullptr,
03819     0, NI_NUMERICHOST) {
03820     return false;
03821 }
03822
03823 ip = ipstr.data();
03824 return true;
03825 }
```

Граф вызова функции:



5.3.2.25 `get_local_ip_and_port()`

```

void httpplib::detail::get_local_ip_and_port (
    socket_t sock,
    std::string & ip,
    int & port) [inline]
```

См. определение в файле `httpplib.h` строка 3827

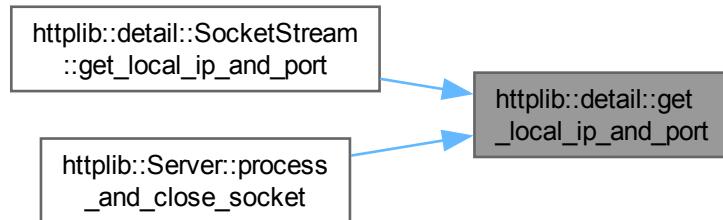
```

03827
03828 struct sockaddr_storage addr;
03829 socklen_t addr_len = sizeof(addr);
03830 if (!getsockname(sock, reinterpret_cast<struct sockaddr*>(&addr),
03831 &addr_len)) {
03832     get_ip_and_port(addr, addr_len, ip, port);
03833 }
03834 }
```

Граф вызовов:



Граф вызова функции:



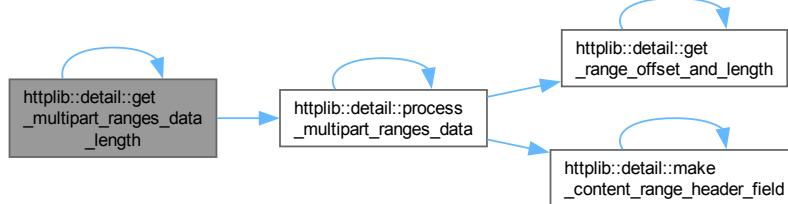
5.3.2.26 `get_multipart_ranges_data_length()`

```
size_t httpplib::detail::get_multipart_ranges_data_length (
    const Request & req,
    const std::string & boundary,
    const std::string & content_type,
    size_t content_length) [inline]
```

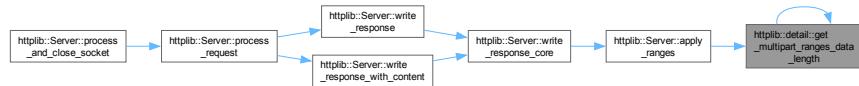
См. определение в файле `httpplib.h` строка 5434

```
05437     size_t data_length = 0;
05438
05439
05440     process_multipart_ranges_data(
05441         req, boundary, content_type, content_length,
05442         [&](const std::string &token) { data_length += token.size(); },
05443         [&](const std::string &token) { data_length += token.size(); },
05444         [&](size_t /*offset*/, size_t length) {
05445             data_length += length;
05446             return true;
05447         });
05448
05449     return data_length;
05450 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.27 `get_range_offset_and_length()`

```
std::pair< size_t, size_t > httpplib::detail::get_range_offset_and_length (
    Range r,
    size_t content_length) [inline]
```

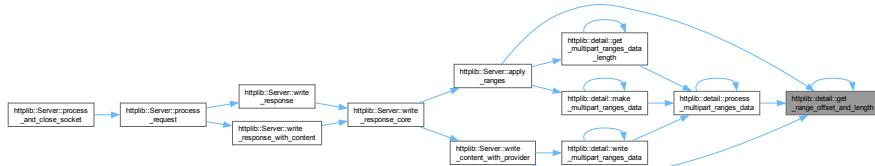
См. определение в файле `httpplib.h` строка 5358

```
05358 {
05359     assert(r.first != -1 && r.second != -1);
05360     assert(0 <= r.first && r.first < static_cast<ssize_t>(content_length));
05361     assert(r.first <= r.second &&
05362             r.second < static_cast<ssize_t>(content_length));
05363     (void)(content_length);
05364     return std::make_pair(r.first, static_cast<size_t>(r.second - r.first) + 1);
05365 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.28 `get_remote_ip_and_port()`

```
void httpplib::detail::get_remote_ip_and_port (
    socket_t sock,
    std::string & ip,
    int & port) [inline]
```

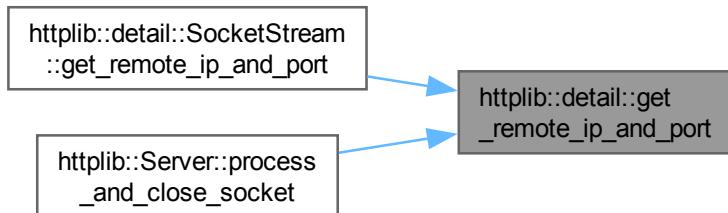
См. определение в файле [httpplib.h](#) строка 3836

```
03836     {
03837     struct sockaddr_storage addr;
03838     socklen_t addr_len = sizeof(addr);
03839
03840     if (!getpeername(sock, reinterpret_cast<struct sockaddr *>(&addr),
03841                      &addr_len)) {
03842 #ifndef _WIN32
03843     if (addr.ss_family == AF_UNIX) {
03844 #if defined(_linux__)
03845         struct ucred ucred;
03846         socklen_t len = sizeof(ucred);
03847         if (getsockopt(sock, SOL_SOCKET, SO_PEERCREDS, &ucred, &len) == 0) {
03848             port = ucred.pid;
03849         }
03850 #elif defined(SOL_LOCAL) && defined(SO_PEERPID) // __APPLE__
03851         pid_t pid;
03852         socklen_t len = sizeof(pid);
03853         if (getsockopt(sock, SOL_LOCAL, SO_PEERPID, &pid, &len) == 0) {
03854             port = pid;
03855         }
03856 #endif
03857     return;
03858 }
03859 #endif
03860     get_ip_and_port(addr, addr_len, ip, port);
03861 }
```

Граф вызовов:



Граф вызова функции:



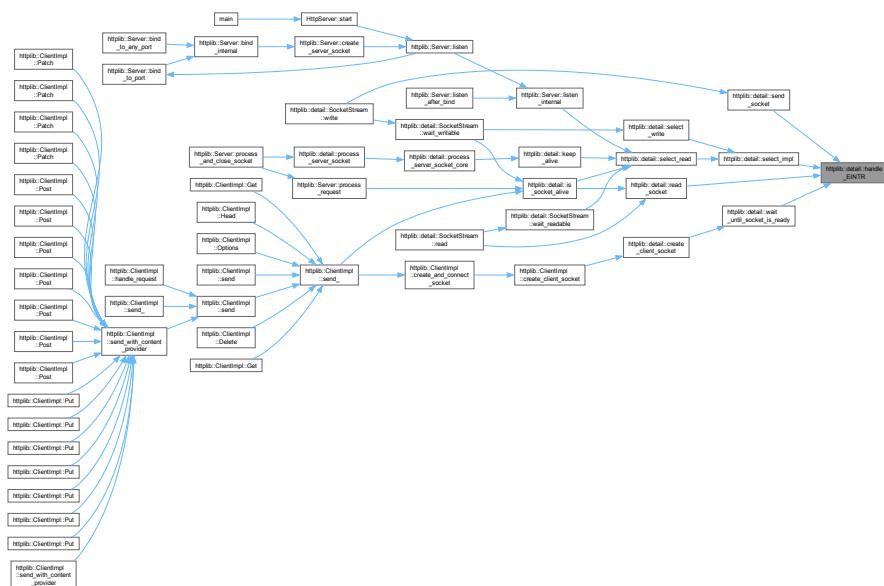
5.3.2.29 handle_EINTR()

```
template<typename T>
ssize_t httplib::detail::handle_EINTR (
    T fn) [inline]
```

См. определение в файле `httpplib.h` строка `3244`

```
03244
03245     ssize_t res = 0;
03246     while (true) {
03247         res = fn();
03248         if (res < 0 && errno == EINTR) {
03249             std::this_thread::sleep_for(std::chrono::microseconds{1});
03250             continue;
03251         }
03252         break;
03253     }
03254     return res;
03255 }
```

Граф вызова функции:



5.3.2.30 has_crlf()

```
bool httplib::detail::has_crlf (
```

См. определение в файле `httpplib.h` строка 5481

```
05481
05482     auto p = s.c_.str();
05483     while (*p) {
05484         if (*p == '\r' || *p == '\n') { return true; }
05485         p++;
05486     }
05487     return false;
05488 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.31 `has_header()`

```
bool httpplib::detail::has_header (
    const Headers & headers,
    const std::string & key) [inline]
```

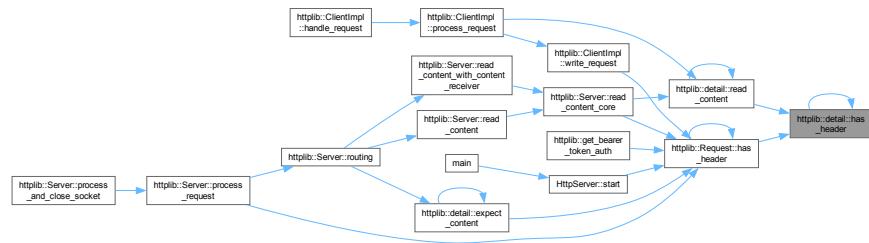
См. определение в файле [httpplib.h](#) строка 4262

```
04262
04263     return headers.find(key) != headers.end();
04264 }
```

Граф вызовов:



Граф вызова функции:



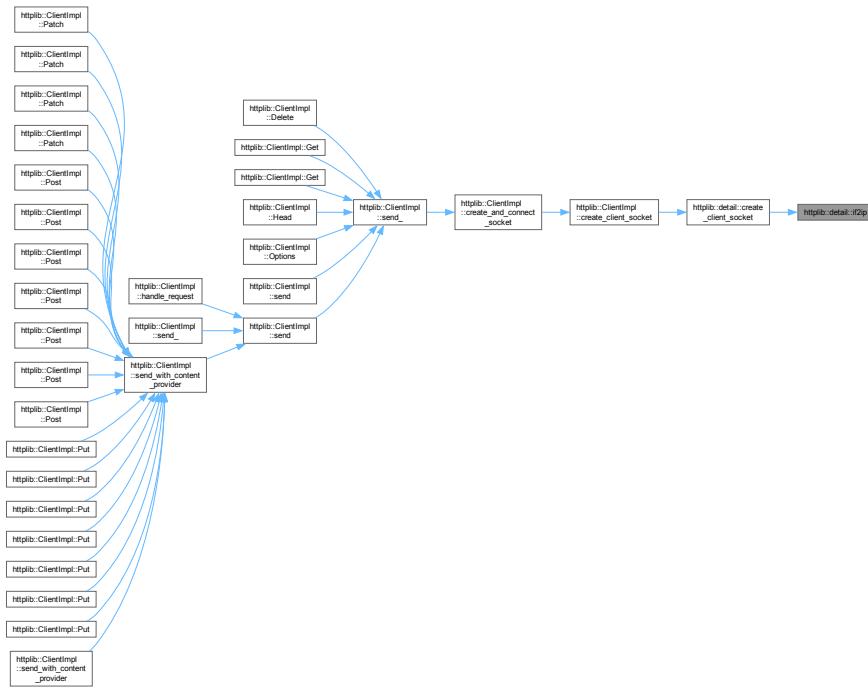
5.3.2.32 `if2ip()`

```
std::string httpplib::detail::if2ip (
    int address_family,
    const std::string & ifn) [inline]
```

См. определение в файле [httpplib.h](#) строка 3709

```
03709
03710     struct ifaddrs *ifap;
03711     getifaddrs(&ifap);
03712     auto se = detail::scope_exit([&] { freeifaddrs(ifap); });
03713
03714     std::string addr_candidate;
03715     for (auto ifa = ifap; ifa; ifa = ifa->ifa_next) {
03716         if (ifa->ifa_addr && ifn == ifa->ifa_name &&
03717             (AF_UNSPEC == address_family ||
03718             ifa->ifa_addr->sa_family == address_family)) {
03719             if (ifa->ifa_addr->sa_family == AF_INET) {
03720                 auto sa = reinterpret_cast<struct sockaddr_in*>(ifa->ifa_addr);
03721                 char buf[INET_ADDRSTRLEN];
03722                 if (inet_ntop(AF_INET, &sa->sin_addr, buf, INET_ADDRSTRLEN)) {
03723                     return std::string(buf, INET_ADDRSTRLEN);
03724                 }
03725             } else if (ifa->ifa_addr->sa_family == AF_INET6) {
03726                 auto sa = reinterpret_cast<struct sockaddr_in6*>(ifa->ifa_addr);
03727                 if (!IN6_IS_ADDR_LINKLOCAL(&sa->sin6_addr)) {
03728                     char buf[INET6_ADDRSTRLEN] = {};
03729                     if (inet_ntop(AF_INET6, &sa->sin6_addr, buf, INET6_ADDRSTRLEN)) {
03730                         // equivalent to mac's IN6_IS_ADDR_UNIQUE_LOCAL
03731                         auto s6_addr_head = sa->sin6_addr.s6_addr[0];
03732                         if (s6_addr_head == 0xfc || s6_addr_head == 0xfd) {
03733                             addr_candidate = std::string(buf, INET6_ADDRSTRLEN);
03734                         } else {
03735                             return std::string(buf, INET6_ADDRSTRLEN);
03736                         }
03737                     }
03738                 }
03739             }
03740         }
03741     }
03742     return addr_candidate;
03743 }
```

Граф вызова функции:



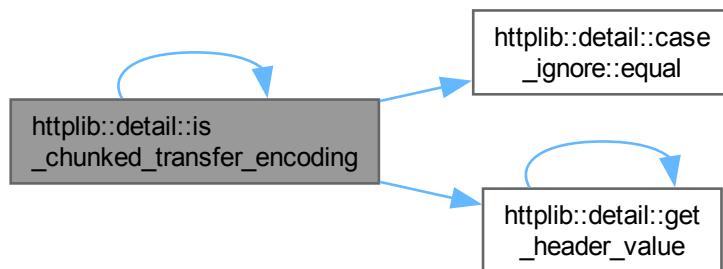
5.3.2.33 is_chunked_transfer_encoding()

```
bool httplib::detail::is_chunked_transfer_encoding (
    const Headers & headers) [inline]
```

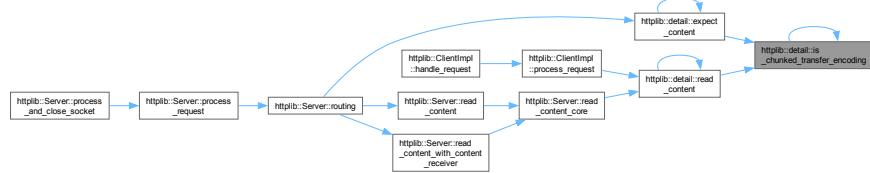
См. определение в файле [httplib.h](#) строка 4477

```
04477
04478     return case_ignore::equal(
04479         get_header_value(headers, "Transfer-Encoding", "", 0), "chunked");
04480 }
```

Граф вызовов:



Граф вызова функции:



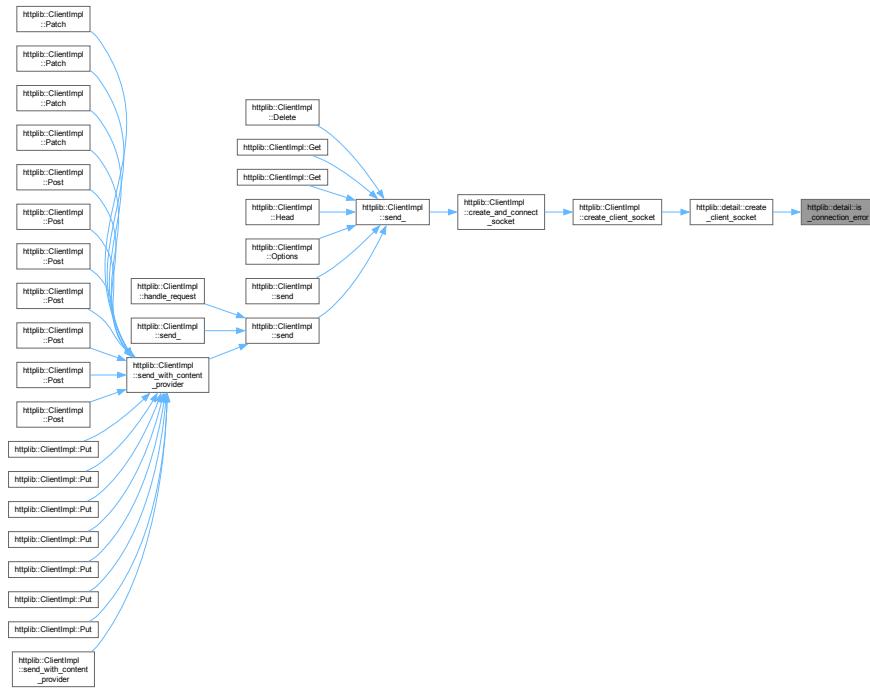
5.3.2.34 is_connection_error()

bool httpplib::detail::is_connection_error () [inline]

См. определение в файле [httplib.h](#) строка 3672

```
03672 {  
03673 #ifndef _WIN32  
03674     return WSAGetLastError() != WSAEWOULDBLOCK;  
03675 #else  
03676     return errno != EINPROGRESS;  
03677 #endif  
03678 }
```

Граф вызова функции:



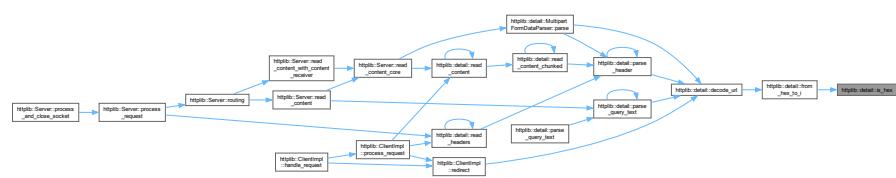
5.3.2.35 is_hex()

```
bool httplib::detail::is_hex (
```

См. определение в файле `httpplib.h` строка 2705

```
02705 {  
02706 if (0x20 <= c && isdigit(c)) {  
02707 v = c - '0';  
02708 return true;  
02709 } else if ('A' <= c && c <= 'F') {  
02710 v = c - 'A' + 10;  
02711 return true;  
02712 } else if ('a' <= c && c <= 'f') {  
02713 v = c - 'a' + 10;  
02714 return true;  
02715 }  
02716 return false;  
02717 }
```

Граф вызова функции:



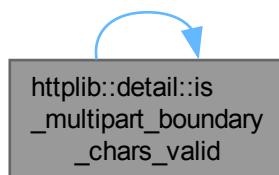
5.3.2.36 is_multipart_boundary_chars_valid()

```
bool httplib::detail::is_multipart_boundary_chars_valid (
```

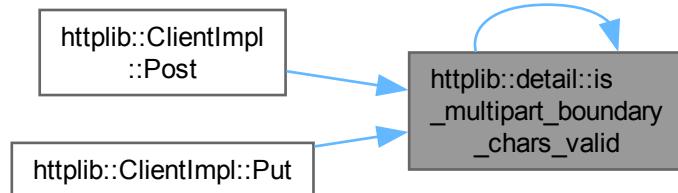
См. определение в файле `httpplib.h` строка 5234

```
05234
05235     auto valid = true;
05236     for (size_t i = 0; i < boundary.size(); i++) {
05237         auto c = boundary[i];
05238         if (!std::isalnum(c) && c != '-' && c != '_') {
05239             valid = false;
05240             break;
05241         }
05242     }
05243     return valid;
05244 }
```

Граф вызовов:



Граф вызова функции:



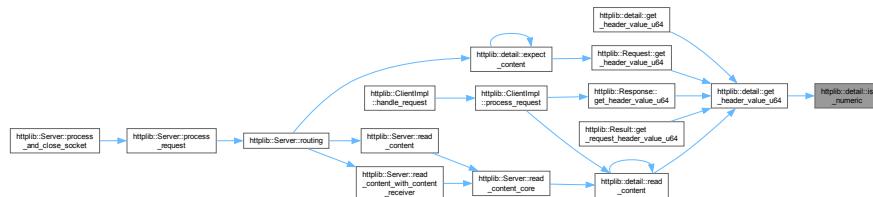
5.3.2.37 `is_numeric()`

```
bool httpplib::detail::is_numeric (
    const std::string & str) [inline]
```

См. определение в файле [httpplib.h](#) строка 2064

```
02064     {
02065         return !str.empty() && std::all_of(str.begin(), str.end(), ::isdigit);
02066     }
```

Граф вызова функции:



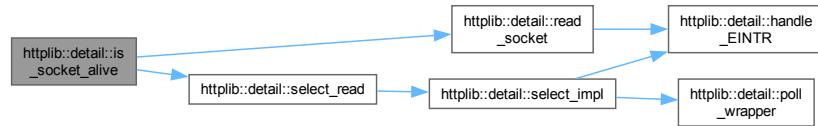
5.3.2.38 `is_socket_alive()`

```
bool httpplib::detail::is_socket_alive (
    socket_t sock) [inline]
```

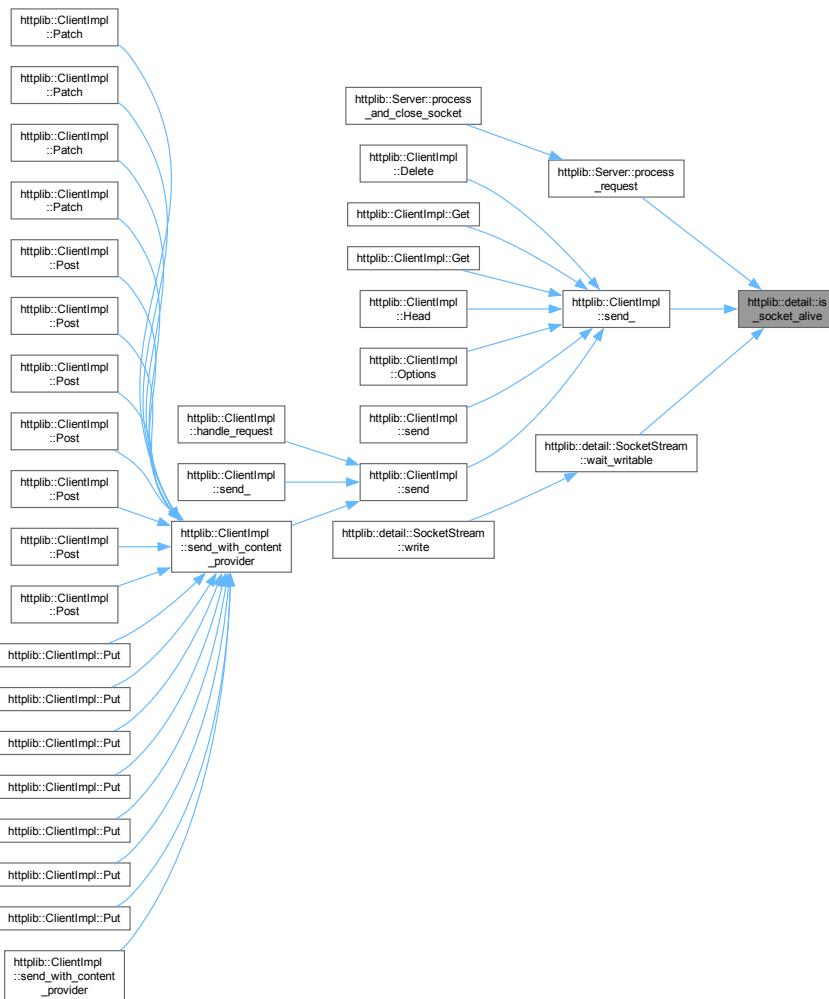
См. определение в файле [httpplib.h](#) строка 3334

```
03334     {
03335         const auto val = detail::select_read(sock, 0, 0);
03336         if (val == 0) {
03337             return true;
03338         } else if (val < 0 && errno == EBADF) {
03339             return false;
03340         }
03341         char buf[1];
03342         return detail::read_socket(sock, &buf[0], sizeof(buf), MSG_PEEK) > 0;
03343     }
```

Граф вызовов:



Граф вызова функции:



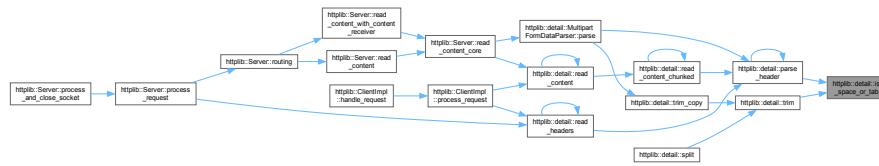
5.3.2.39 `is_space_or_tab()`

```
bool httpplib::detail::is_space_or_tab (
    char c) [inline]
```

См. определение в файле [httpplib.h](#) строка 2962

```
02962 { return c == ' ' || c == '\t'; }
```

Граф вызова функции:



5.3.2.40 `is_valid_path()`

```
bool httpplib::detail::is_valid_path (
    const std::string & path) [inline]
```

См. определение в файле [httpplib.h](#) строка 2808

```
02808 {
02809     size_t level = 0;
02810     size_t i = 0;
02811
02812     // Skip slash
02813     while (i < path.size() && path[i] == '/') {
02814         i++;
02815     }
02816
02817     while (i < path.size()) {
02818         // Read component
02819         auto beg = i;
02820         while (i < path.size() && path[i] != '/') {
02821             if (path[i] == '\0') {
02822                 return false;
02823             } else if (path[i] == '\\') {
02824                 return false;
02825             }
02826             i++;
02827         }
02828
02829         auto len = i - beg;
02830         assert(len > 0);
02831
02832         if (!path.compare(beg, len, "."))
02833             ;
02834         } else if (!path.compare(beg, len, "..")) {
02835             if (level == 0) { return false; }
02836             level--;
02837         } else {
02838             level++;
02839         }
02840
02841     // Skip slash
02842     while (i < path.size() && path[i] == '/') {
02843         i++;
02844     }
02845 }
02846
02847 return true;
02848 }
```

Граф вызова функции:



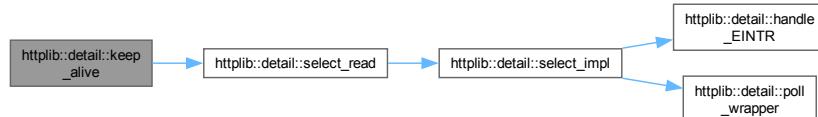
5.3.2.41 `keep_alive()`

```
bool httpplib::detail::keep_alive (
    const std::atomic< socket_t > & svr_sock,
    socket_t sock,
    time_t keep_alive_timeout_sec) [inline]
```

См. определение в файле `httpplib.h` строка 3413

```
03414     {
03415     using namespace std::chrono;
03416
03417     const auto interval_usec =
03418         CPPHTTPPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND;
03419
03420     // Avoid expensive `steady_clock::now()` call for the first time
03421     if (select_read(sock, 0, interval_usec) > 0) { return true; }
03422
03423     const auto start = steady_clock::now() - microseconds{interval_usec};
03424     const auto timeout = seconds{keep_alive_timeout_sec};
03425
03426     while (true) {
03427         if (svr_sock == INVALID_SOCKET) {
03428             break; // Server socket is closed
03429         }
03430
03431         auto val = select_read(sock, 0, interval_usec);
03432         if (val < 0) {
03433             break; // Ssocket error
03434         } else if (val == 0) {
03435             if (steady_clock::now() - start > timeout) {
03436                 break; // Timeout
03437             }
03438         } else {
03439             return true; // Ready for read
03440         }
03441     }
03442
03443     return false;
03444 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.42 `make_content_range_header_field()`

```
std::string httpplib::detail::make_content_range_header_field (
    const std::pair< size_t, size_t > & offset_and_length,
    size_t content_length) [inline]
```

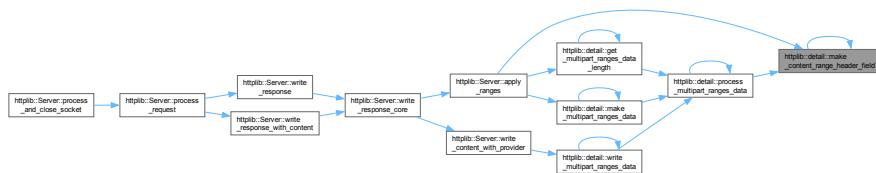
См. определение в файле `httpplib.h` строка 5367

```
05368     auto st = offset_and_length.first;
05369     auto ed = st + offset_and_length.second - 1;
05370
05371     std::string field = "bytes ";
05372     field += std::to_string(st);
05373     field += "_";
05374     field += std::to_string(ed);
05375     field += "/";
05376     field += std::to_string(content_length);
05377
05378     return field;
05379 }
```

Граф вызовов:



Граф вызова функции:



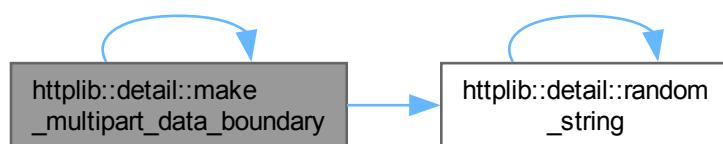
5.3.2.43 `make_multipart_data_boundary()`

`std::string httpplib::detail::make_multipart_data_boundary () [inline]`

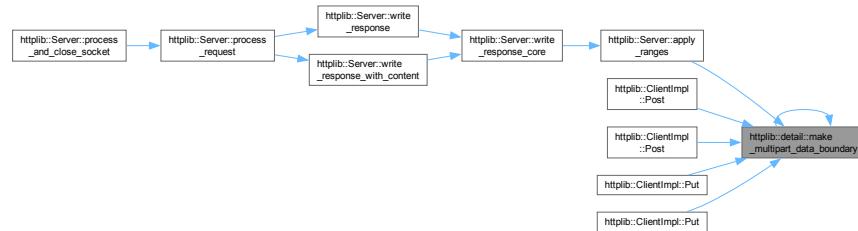
См. определение в файле `httpplib.h` строка 5230

```
05230
05231     return "--cpp-httpplib-multipart-data--" + detail::random_string(16);
05232 }
```

Граф вызовов:



Граф вызова функции:



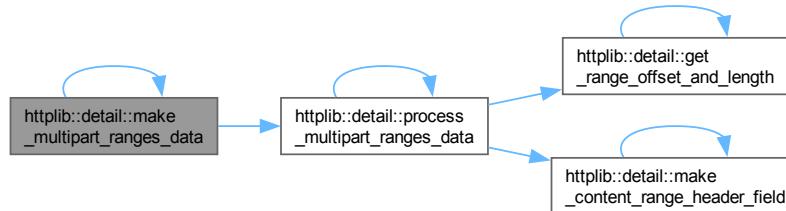
5.3.2.44 `make_multipart_ranges_data()`

```
void httpplib::detail::make_multipart_ranges_data (
    const Request & req,
    Response & res,
    const std::string & boundary,
    const std::string & content_type,
    size_t content_length,
    std::string & data) [inline]
```

См. определение в файле `httpplib.h` строка 5418

```
05422 {
05423     process_multipart_ranges_data(
05424         req, boundary, content_type, content_length,
05425         [&](const std::string &token) { data += token; },
05426         [&](const std::string &token) { data += token; },
05427         [&](size_t offset, size_t length) {
05428             assert(offset + length <= content_length);
05429             data += res.body.substr(offset, length);
05430             return true;
05431         });
05432 }
```

Граф вызовов:



Граф вызова функции:



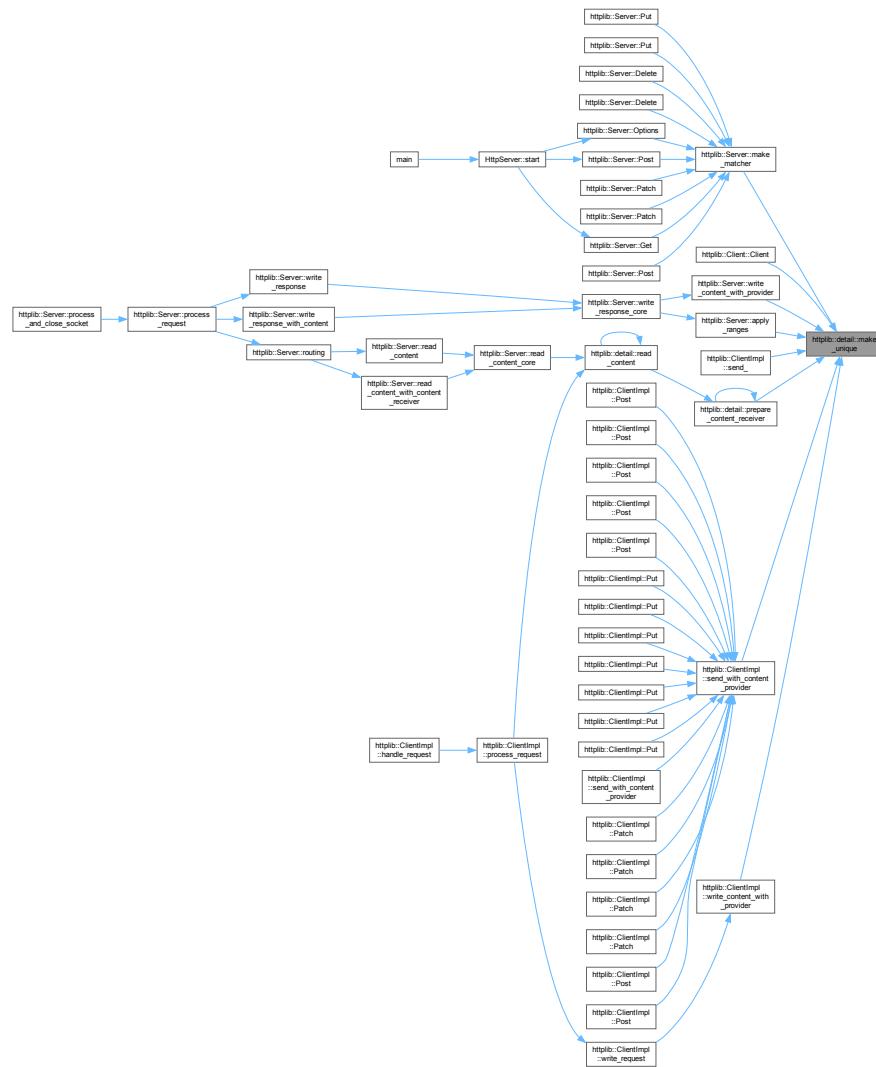
5.3.2.45 make_unique() [1/2]

```
template<class T, class... Args>
std::enable_if<!std::is_array< T >::value, std::unique_ptr< T > >::type httpplib::detail::make_unique (
    Args &&... args)
```

См. определение в файле `httpplib.h` строка 339

```
00339 {  
00340     return std::unique_ptr<T>(new T(std::forward<Args>(args)...));  
00341 }
```

Граф вызова функции:



5.3.2.46 make_unique() [2/2]

```
template<class T>
std::enable_if< std::is_array< T >::value, std::unique_ptr< T > >::type httpplib::detail::make_unique (
    std::size_t n)
```

См. определение в файле `httpplib.h` строка 345

```
00345         {
00346     typedef typename std::remove_extent<T>::type RT;
00347     return std::unique_ptr<T>(new RT[n]);
00348 }
```

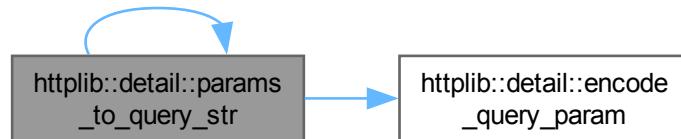
5.3.2.47 params_to_query_str()

```
std::string httplib::detail::params_to_query_str (
    const Params & params) [inline]
```

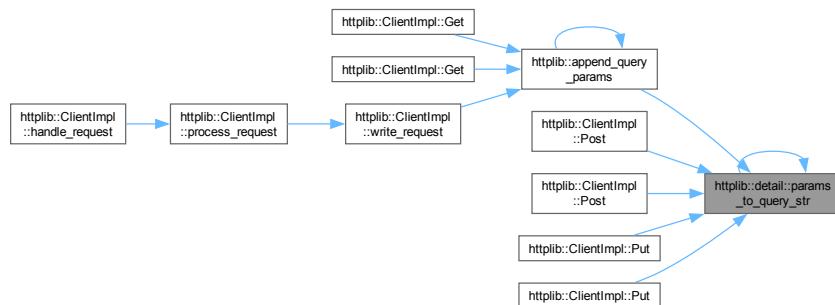
См. определение в файле [httplib.h](#) строка 4839

```
04839
04840 std::string query;
04841
04842 for (auto it = params.begin(); it != params.end(); ++it) {
04843     if (it != params.begin()) { query += "&"; }
04844     query += it->first;
04845     query += "=";
04846     query += encode_query_param(it->second);
04847 }
04848 return query;
04849 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.48 parse_disposition_params()

```
void httplib::detail::parse_disposition_params (
    const std::string & s,
    Params & params) [inline]
```

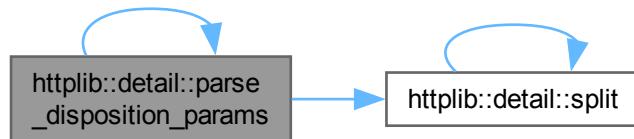
См. определение в файле [httplib.h](#) строка 4889

```
04889
04890 std::set<std::string> cache;
```

```

04891 split(s.data(), s.data() + s.size(), ';, [&](const char *b, const char *e) {
04892     std::string kv(b, e);
04893     if (cache.find(kv) != cache.end()) { return; }
04894     cache.insert(kv);
04895
04896     std::string key;
04897     std::string val;
04898     split(b, e, '=', [&](const char *b2, const char *e2) {
04899         if (key.empty()) {
04900             key.assign(b2, e2);
04901         } else {
04902             val.assign(b2, e2);
04903         }
04904     });
04905
04906     if (!key.empty()) {
04907         params.emplace(trim_double_quotes_copy((key)),
04908                         trim_double_quotes_copy((val)));
04909     }
04910 };
04911 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.49 `parse_header()`

```

template<typename T>
bool httpplib::detail::parse_header(
    const char * beg,
    const char * end,
    T fn) [inline]
```

См. определение в файле `httpplib.h` строка [4277](#)

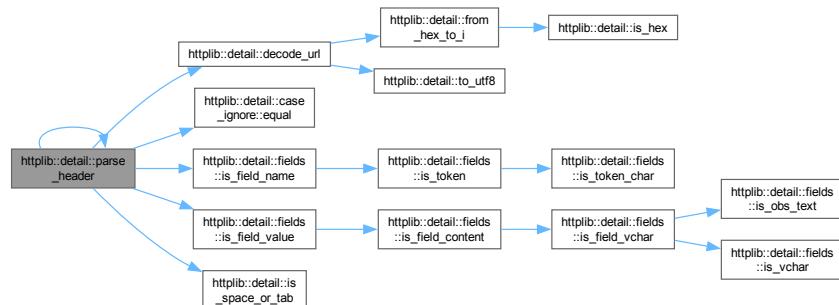
```

04277                                     {
04278     // Skip trailing spaces and tabs.
04279     while (beg < end && is_space_or_tab(end[-1])) {
04280         end--;
04281     }
04282
04283     auto p = beg;
04284     while (p < end && *p != '.') {
04285         p++;
04286     }
```

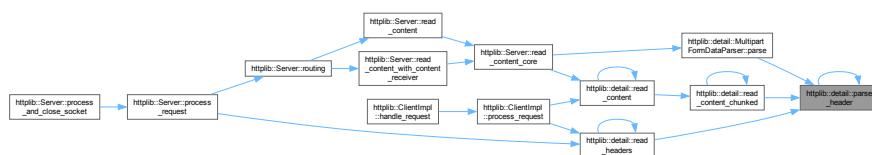
```

04287
04288     auto name = std::string(beg, p);
04289     if (!detail::fields::is_field_name(name)) { return false; }
04290
04291     if (p == end) { return false; }
04292
04293     auto key_end = p;
04294
04295     if (*p++ != ':') { return false; }
04296
04297     while (p < end && is_space_or_tab(*p)) {
04298         p++;
04299     }
04300
04301     if (p <= end) {
04302         auto key_len = key_end - beg;
04303         if (!key_len) { return false; }
04304
04305         auto key = std::string(beg, key_end);
04306         auto val = std::string(p, end);
04307
04308         if (!detail::fields::is_field_value(val)) { return false; }
04309
04310         if (case_ignore::equal(key, "Location") ||
04311             case_ignore::equal(key, "Referer")) {
04312             fn(key, val);
04313         } else {
04314             fn(key, decode_url(val, false));
04315         }
04316
04317     return true;
04318 }
04319
04320 return false;
04321 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.50 parse_multipart_boundary()

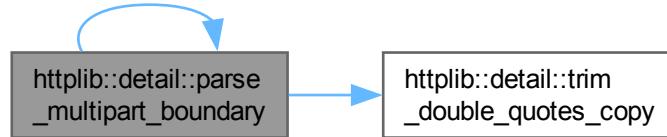
```

bool http://detail::parse_multipart_boundary (
    const std::string & content_type,
    std::string & boundary) [inline]
```

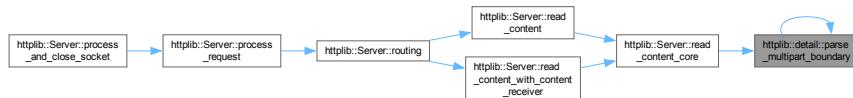
См. определение в файле `httpplib.h` строка 4878

```
04879     auto boundary_keyword = "boundary=\"";
04880     auto pos = content_type.find(boundary_keyword);
04881     if (pos == std::string::npos) { return false; }
04882     auto end = content_type.find(';', pos);
04883     auto beg = pos + strlen(boundary_keyword);
04884     boundary = trim_double_quotes_copy(content_type.substr(beg, end - beg));
04885     return !boundary.empty();
04886 }
04887 }
```

Граф вызовов:



Граф вызова функции:



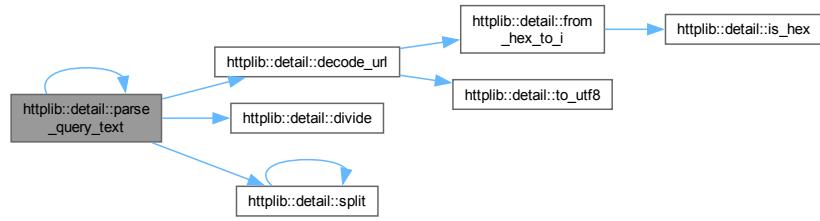
5.3.2.51 parse_query_text() [1/2]

```
void httpplib::detail::parse_query_text (
    const char * data,
    std::size_t size,
    Params & params) [inline]
```

См. определение в файле `httpplib.h` строка 4851

```
04852     {
04853     std::set<std::string> cache;
04854     split(data, data + size, '&', [&](const char *b, const char *e) {
04855         std::string kv(b, e);
04856         if (cache.find(kv) != cache.end()) { return; }
04857         cache.insert(std::move(kv));
04858
04859         std::string key;
04860         std::string val;
04861         divide(b, static_cast<std::size_t>(e - b), '=',
04862             [&](const char *lhs_data, std::size_t lhs_size, const char *rhs_data,
04863                 std::size_t rhs_size) {
04864                 key.assign(lhs_data, lhs_size);
04865                 val.assign(rhs_data, rhs_size);
04866             });
04867
04868         if (!key.empty()) {
04869             params.emplace(decode_url(key, true), decode_url(val, true));
04870         }
04871     });
04872 }
```

Граф вызовов:



Граф вызова функции:



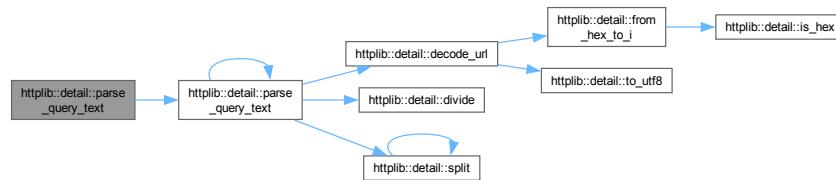
5.3.2.52 `parse_query_text()` [2/2]

```
void httpplib::detail::parse_query_text (
    const std::string & s,
    Params & params) [inline]
```

См. определение в файле [httpplib.h](#) строка 4874

```
04874
04875  parse_query_text(s.data(), s.size(), params);
04876 }
```

Граф вызовов:



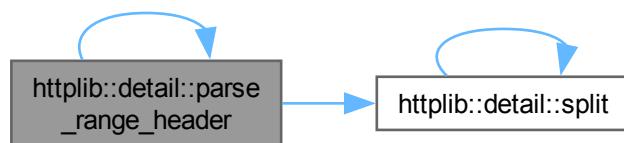
5.3.2.53 parse_range_header()

```
bool httpplib::detail::parse_range_header (
    const std::string & s,
    Ranges & ranges) [inline]
```

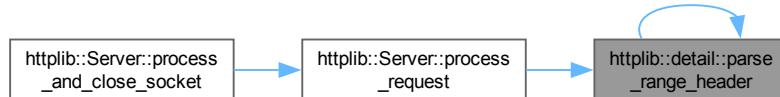
См. определение в файле [httpplib.h](#) строка 4916

```
04916                                     {
04917 #endif
04918     auto is_valid = [](<const std::string &str) {
04919         return std::all_of(str.cbegin(), str.cend(),
04920                             [](unsigned char c) { return std::isdigit(c); });
04921     };
04922
04923     if (s.size() > 7 && s.compare(0, 6, "bytes=") == 0) {
04924         const auto pos = static_cast<size_t>(6);
04925         const auto len = static_cast<size_t>(s.size() - 6);
04926         auto all_valid_ranges = true;
04927         split(&s[pos], &s[pos + len], ',', [&](const char *b, const char *e) {
04928             if (!all_valid_ranges) { return; }
04929
04930             const auto it = std::find(b, e, '-');
04931             if (it == e) {
04932                 all_valid_ranges = false;
04933                 return;
04934             }
04935
04936             const auto lhs = std::string(b, it);
04937             const auto rhs = std::string(it + 1, e);
04938             if (!is_valid(lhs) || !is_valid(rhs)) {
04939                 all_valid_ranges = false;
04940                 return;
04941             }
04942
04943             const auto first =
04944                 static_cast<ssize_t>(lhs.empty() ? -1 : std::stoll(lhs));
04945             const auto last =
04946                 static_cast<ssize_t>(rhs.empty() ? -1 : std::stoll(rhs));
04947             if ((first == -1 && last == -1) ||
04948                 (first != -1 && last != -1 && first > last)) {
04949                 all_valid_ranges = false;
04950                 return;
04951             }
04952
04953             ranges.emplace_back(first, last);
04954         });
04955         return all_valid_ranges && !ranges.empty();
04956     }
04957     return false;
04958 #ifdef CPPHTTPLIB_NO_EXCEPTIONS
04959 }
04960 #else
04961 } catch (...) { return false; }
```

Граф вызовов:



Граф вызова функции:



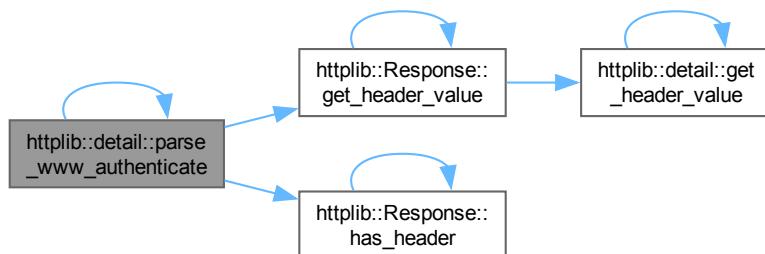
5.3.2.54 `parse_www_authenticate()`

```
bool httpplib::detail::parse_www_authenticate (
    const Response & res,
    std::map< std::string, std::string > & auth,
    bool is_proxy) [inline]
```

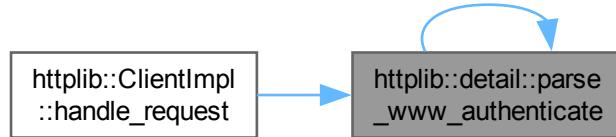
См. определение в файле `httpplib.h` строка 5730

```
05732     {
05733     auto auth_key = is_proxy ? "Proxy-Authenticate" : "WWW-Authenticate";
05734     if (res.has_header(auth_key)) {
05735         thread_local auto re =
05736             std::regex(R"~((?:(?:\s*)?(.+?)=(?:"(.?")|([^\n,]*))))~");
05737         auto s = res.get_header_value(auth_key);
05738         auto pos = s.find(' ');
05739         if (pos != std::string::npos) {
05740             auto type = s.substr(0, pos);
05741             if (type == "Basic") {
05742                 return false;
05743             } else if (type == "Digest") {
05744                 s = s.substr(pos + 1);
05745                 auto beg = std::sregex_iterator(s.begin(), s.end(), re);
05746                 for (auto i = beg; i != std::sregex_iterator(); ++i) {
05747                     const auto &m = *i;
05748                     auto key = s.substr(static_cast<size_t>(m.position(1)),
05749                                         static_cast<size_t>(m.length(1)));
05750                     auto val = m.length(2) > 0
05751                         ? s.substr(static_cast<size_t>(m.position(2)),
05752                                         static_cast<size_t>(m.length(2)))
05753                         : s.substr(static_cast<size_t>(m.position(3)),
05754                                         static_cast<size_t>(m.length(3)));
05755                     auth[key] = val;
05756                 }
05757                 return true;
05758             }
05759         }
05760     }
05761     return false;
05762 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.55 poll_wrapper()

```

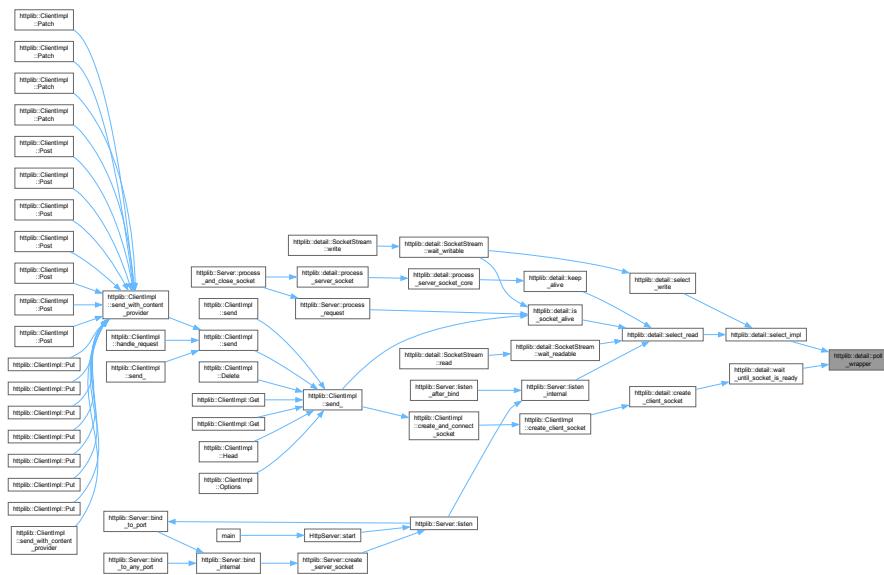
int httpplib::detail::poll_wrapper (
    struct pollfd * fds,
    nfds_t nfds,
    int timeout) [inline]
  
```

См. определение в файле `httpplib.h` строка 3282

```

03282 {
03283 #ifdef _WIN32
03284     return ::WSAPoll(fds, nfds, timeout);
03285 #else
03286     return ::poll(fds, nfds, timeout);
03287 #endif
03288 }
  
```

Граф вызова функции:



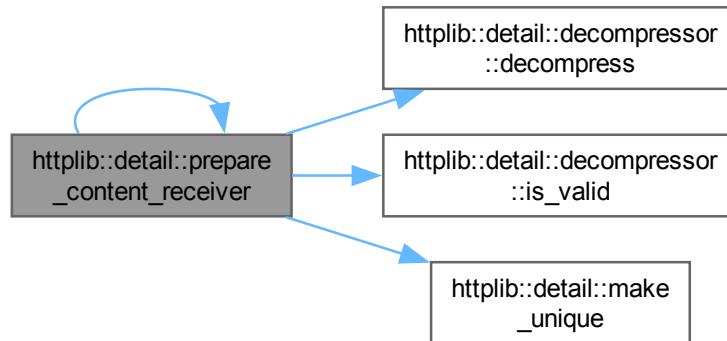
5.3.2.56 prepare_content_receiver()

```
template<typename T, typename U>
bool httpplib::detail::prepare_content_receiver (
    T & x,
    int & status,
    ContentReceiverWithProgress receiver,
    bool decompress,
    U callback)
```

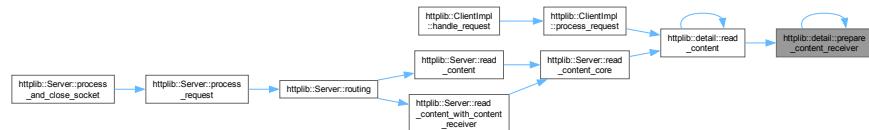
См. определение в файле [httpplib.h](#) строка 4483

```
04485         {
04486     if (decompress) {
04487         std::string encoding = x.get_header_value("Content-Encoding");
04488         std::unique_ptr<decompressor> decompressor;
04489
04490         if (encoding == "gzip" || encoding == "deflate") {
04491 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
04492             decompressor = detail::make_unique<gzip_decompressor>();
04493 #else
04494             status = StatusCode::UnsupportedMediaType_415;
04495             return false;
04496 #endif
04497     } else if (encoding.find("br") != std::string::npos) {
04498 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
04499             decompressor = detail::make_unique<brotli_decompressor>();
04500 #else
04501             status = StatusCode::UnsupportedMediaType_415;
04502             return false;
04503 #endif
04504     } else if (encoding == "zstd") {
04505 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
04506             decompressor = detail::make_unique<zstd_decompressor>();
04507 #else
04508             status = StatusCode::UnsupportedMediaType_415;
04509             return false;
04510 #endif
04511     }
04512
04513     if (decompressor) {
04514         if (decompressor->is_valid()) {
04515             ContentReceiverWithProgress out = [&](const char *buf, size_t n,
04516                                         uint64_t off, uint64_t len) {
04517                 return decompressor->decompress(buf, n,
04518                                                 [&](const char *buf2, size_t n2) {
04519                     return receiver(buf2, n2, off, len);
04520                 });
04521             };
04522             return callback(std::move(out));
04523         } else {
04524             status = StatusCode::InternalServerError_500;
04525             return false;
04526         }
04527     }
04528 }
04529 ContentReceiverWithProgress out = [&](const char *buf, size_t n, uint64_t off,
04530                                         uint64_t len) {
04531     return receiver(buf, n, off, len);
04532 };
04533 };
04534 return callback(std::move(out));
04535 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.57 `process_client_socket()`

```

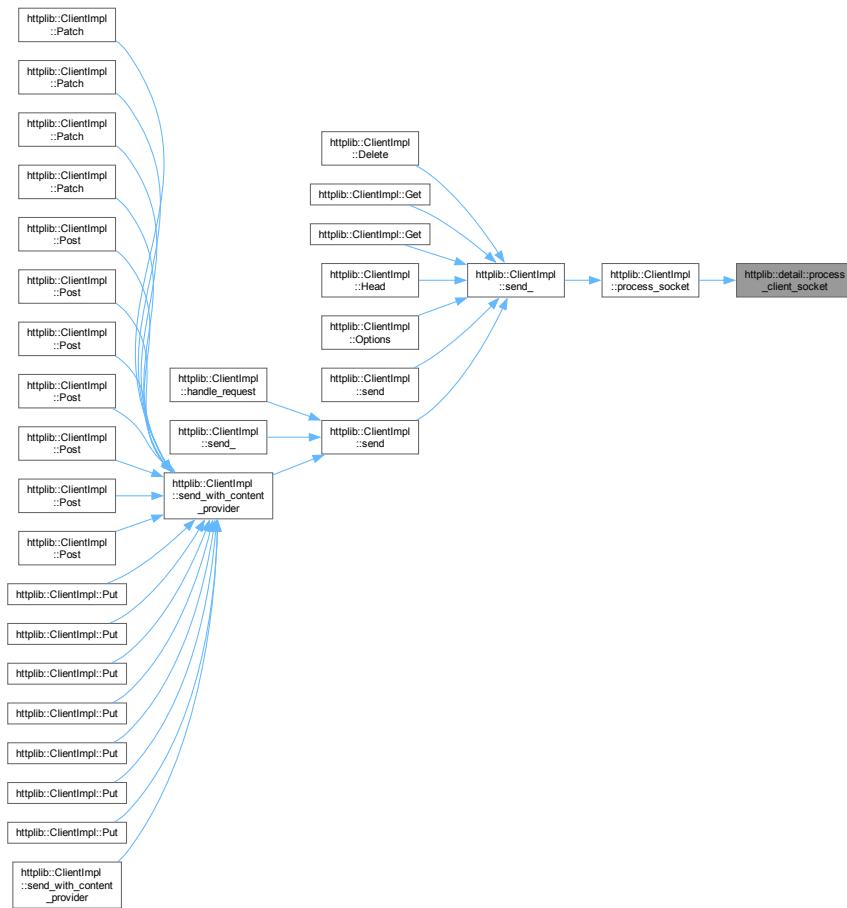
bool httpplib::detail::process_client_socket (
    socket_t sock,
    time_t read_timeout_sec,
    time_t read_timeout_usec,
    time_t write_timeout_sec,
    time_t write_timeout_usec,
    time_t max_timeout_msec,
    std::chrono::time_point<std::chrono::steady_clock> start_time,
    std::function< bool(Stream &) > callback) [inline]
  
```

См. определение в файле [httpplib.h](#) строка 3480

```

03485      {
03486      SocketStream strm(sock, read_timeout_sec, read_timeout_usec,
03487                      write_timeout_sec, write_timeout_usec, max_timeout_msec,
03488                      start_time);
03489      return callback(strm);
03490  }
  
```

Граф вызова функции:



5.3.2.58 process_multipart_ranges_data()

```

template<typename SToken, typename CToken, typename Content>
bool httplib::detail::process_multipart_ranges_data (
    const Request & req,
    const std::string & boundary,
    const std::string & content_type,
    size_t content_length,
    SToken stoken,
    CToken ctoken,
    Content content)
  
```

См. определение в файле [httplib.h](#) строка 5382

```

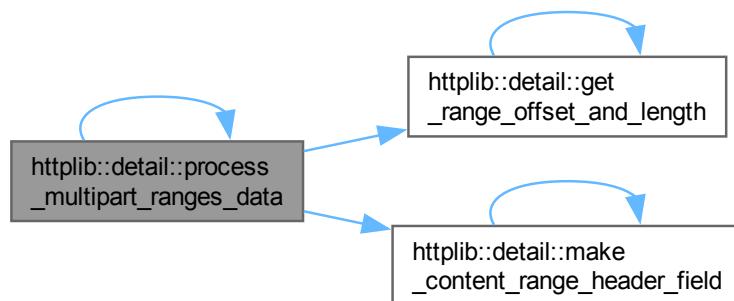
05386
05387   for (size_t i = 0; i < req.ranges.size(); i++) {
05388     ctokn("\r\n");
05389     stoken(boundary);
05390     ctokn("\r\n");
05391     if (!content_type.empty()) {
05392       ctokn("Content-Type: ");
05393       stoken(content_type);
05394       ctokn("\r\n");
05395     }
05396   }
  
```

```

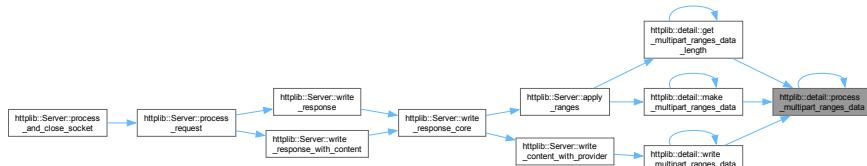
05397 auto offset_and_length =
05398     get_range_offset_and_length(req.ranges[i], content_length);
05399
05400 ctoken("Content-Range: ");
05401 stoken(make_content_range_header_field(offset_and_length, content_length));
05402 ctoken("\r\n");
05403 ctoken("\r\n");
05404
05405 if (!content(offset_and_length.first, offset_and_length.second)) {
05406     return false;
05407 }
05408 ctoken("\r\n");
05409 }
05410
05411 ctoken("--");
05412 stoken(boundary);
05413 ctoken("--");
05414
05415 return true;
05416 }

```

Граф вызовов:



Граф вызова функции:



5.3.2.59 `process_server_socket()`

```

template<typename T>
bool httpplib::detail::process_server_socket (
    const std::atomic<socket_t> & svr_sock,
    socket_t sock,
    size_t keep_alive_max_count,
    time_t keep_alive_timeout_sec,
    time_t read_timeout_sec,

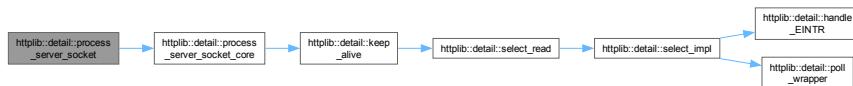
```

```
time_t read_timeout_usec,
time_t write_timeout_sec,
time_t write_timeout_usec,
T callback) [inline]
```

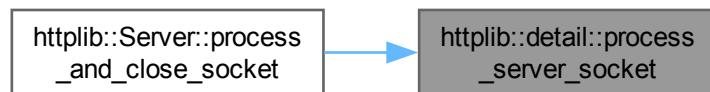
См. определение в файле [httpplib.h](#) строка 3466

```
03470 {
03471     return process_server_socket_core(
03472         svr_sock, sock, keep_alive_max_count, keep_alive_timeout_sec,
03473         [&](bool close_connection, bool &connection_closed) {
03474             SocketStream strm(sock, read_timeout_sec, read_timeout_usec,
03475                             write_timeout_sec, write_timeout_usec);
03476             return callback(strm, close_connection, connection_closed);
03477         });
03478 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.60 `process_server_socket_core()`

```
template<typename T>
bool httpplib::detail::process_server_socket_core (
    const std::atomic<socket_t> & svr_sock,
    socket_t sock,
    size_t keep_alive_max_count,
    time_t keep_alive_timeout_sec,
    T callback) [inline]
```

См. определение в файле [httpplib.h](#) строка 3448

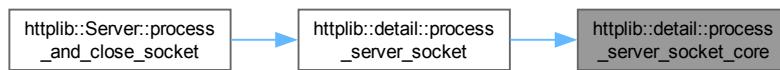
```
03450 {
03451     assert(keep_alive_max_count > 0);
03452     auto ret = false;
03453     auto count = keep_alive_max_count;
03454     while (count > 0 && keep_alive(svr_sock, sock, keep_alive_timeout_sec)) {
03455         auto close_connection = count == 1;
03456         auto connection_closed = false;
03457         ret = callback(close_connection, connection_closed);
03458         if (!ret || connection_closed) { break; }
03459         count--;
03460     }
03461     return ret;
```

```
03462 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.61 `random_string()`

```
std::string httpplib::detail::random_string (
    size_t length) [inline]
```

См. определение в файле [httpplib.h](#) строка 5209

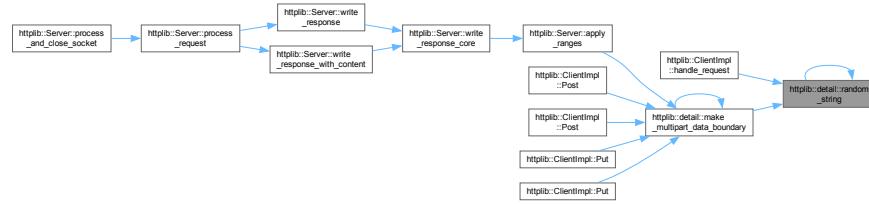
```

05209
05210     constexpr const char data[] =
05211         "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
05212
05213     thread_local auto engine([]() {
05214         // std::random_device might actually be deterministic on some
05215         // platforms, but due to lack of support in the c++ standard library,
05216         // doing better requires either some ugly hacks or breaking portability.
05217         std::random_device seed_gen;
05218         // Request 128 bits of entropy for initialization
05219         std::seed_seq seed_sequence{seed_gen(), seed_gen(), seed_gen(), seed_gen()};
05220         return std::mt19937(seed_sequence);
05221     }());
05222
05223     std::string result;
05224     for (size_t i = 0; i < length; i++) {
05225         result += data[engine() % (sizeof(data) - 1)];
05226     }
05227     return result;
05228 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.62 `range_error()`

```
bool httpplib::detail::range_error (
    Request & req,
    Response & res) [inline]
```

См. определение в файле `httpplib.h` строка 5291

```

05291     {
05292     if (!req.ranges.empty() && 200 <= res.status && res.status < 300) {
05293         ssizet content_len = static_cast<ssizet>(
05294             res.content_length_ ? res.content_length_ : res.body.size());
05295
05296         ssizet prev_first_pos = -1;
05297         ssizet prev_last_pos = -1;
05298         size_t overwrapping_count = 0;
05299
05300         // NOTE: The following Range check is based on '14.2. Range' in RFC 9110
05301         // 'HTTP Semantics' to avoid potential denial-of-service attacks.
05302         // https://www.rfc-editor.org/rfc/rfc9110#section-14.2
05303
05304         // Too many ranges
05305         if (req.ranges.size() > CPPHTTPLIB_RANGE_MAX_COUNT) { return true; }
05306
05307         for (auto &r : req.ranges) {
05308             auto &first_pos = r.first;
05309             auto &last_pos = r.second;
05310
05311             if (first_pos == -1 && last_pos == -1) {
05312                 first_pos = 0;
05313                 last_pos = content_len;
05314             }
05315
05316             if (first_pos == -1) {
05317                 first_pos = content_len - last_pos;
05318                 last_pos = content_len - 1;
05319             }
05320
05321             // NOTE: RFC-9110 '14.1.2. Byte Ranges':
05322             // A client can limit the number of bytes requested without knowing the
05323             // size of the selected representation. If the last-pos value is absent,
05324             // or if the value is greater than or equal to the current length of the
05325             // representation data, the byte range is interpreted as the remainder of
05326             // the representation (i.e., the server replaces the value of last-pos
05327             // with a value that is one less than the current length of the selected
05328             // representation).
05329             // https://www.rfc-editor.org/rfc/rfc9110.html#section-14.1.2-6
05330             if (last_pos == -1 || last_pos >= content_len) {
05331                 last_pos = content_len - 1;
05332             }
05333
05334             // Range must be within content length
05335             if (!(0 <= first_pos && first_pos <= last_pos &&
05336                 last_pos <= content_len - 1)) {
05337                 return true;
05338             }
05339
05340             // Ranges must be in ascending order
05341             if (first_pos <= prev_first_pos) { return true; }
05342
05343             // Request must not have more than two overlapping ranges
05344             if (first_pos <= prev_last_pos) {

```

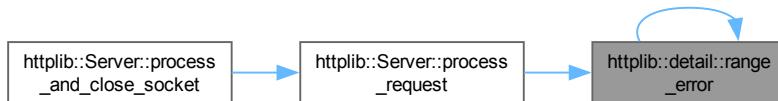
```

05345     overwrapping_count++;
05346     if (overwrapping_count > 2) { return true; }
05347 }
05348
05349     prev_first_pos = (std::max)(prev_first_pos, first_pos);
05350     prev_last_pos = (std::max)(prev_last_pos, last_pos);
05351 }
05352 }
05353
05354 return false;
05355 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.63 `read_content()`

```

template<typename T>
bool httpplib::detail::read_content(
    Stream & strm,
    T & x,
    size_t payload_max_length,
    int & status,
    Progress progress,
    ContentReceiverWithProgress receiver,
    bool decompress)
```

См. определение в файле [httpplib.h](#) строка 4538

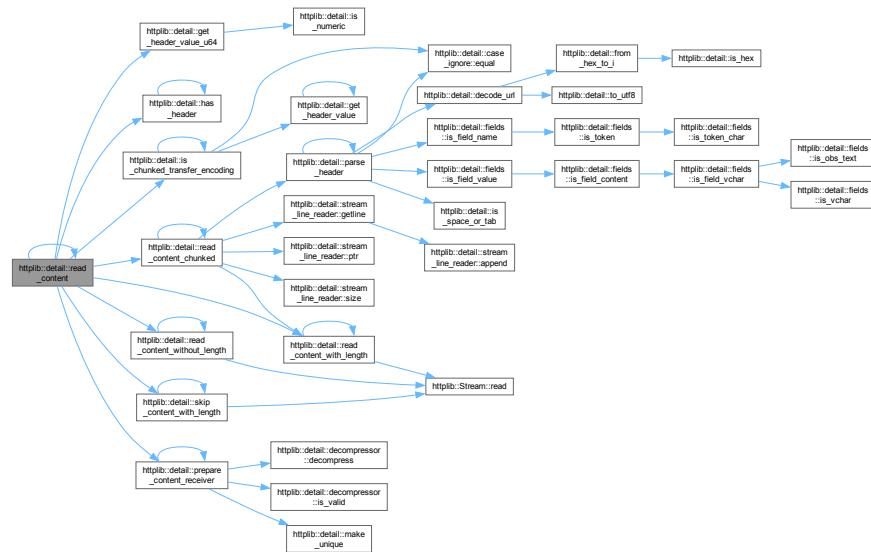
```

04540     {
04541     return prepare_content_receiver(
04542         x, status, std::move(receiver), decompress,
04543         [&](const ContentReceiverWithProgress &out) {
04544             auto ret = true;
04545             auto exceed_payload_max_length = false;
04546
04547             if (is_chunked_transfer_encoding(x.headers)) {
04548                 ret = read_content_chunked(strm, x, out);
04549             }
04550         });
04551     }
04552 }
```

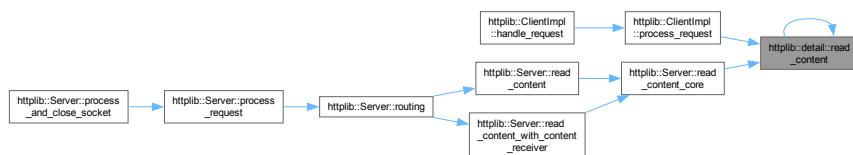
```

04549     } else if (!has_header(x.headers, "Content-Length")) {
04550         ret = read_content_without_length(strm, out);
04551     } else {
04552         auto is_invalid_value = false;
04553         auto len = get_header_value_u64(
04554             x.headers, "Content-Length",
04555             (std::numeric_limits<uint64_t>::max)(), 0, is_invalid_value);
04556
04557         if (is_invalid_value) {
04558             ret = false;
04559         } else if (len > payload_max_length) {
04560             exceed_payload_max_length = true;
04561             skip_content_with_length(strm, len);
04562             ret = false;
04563         } else if (len > 0) {
04564             ret = read_content_with_length(strm, len, std::move(progress), out);
04565         }
04566     }
04567
04568     if (!ret) {
04569         status = exceed_payload_max_length ? StatusCode::PayloadTooLarge_413
04570             : StatusCode::BadRequest_400;
04571     }
04572     return ret;
04573 };
04574 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.64 read_content_chunked()

```

template<typename T>
bool httplib::detail::read_content_chunked (

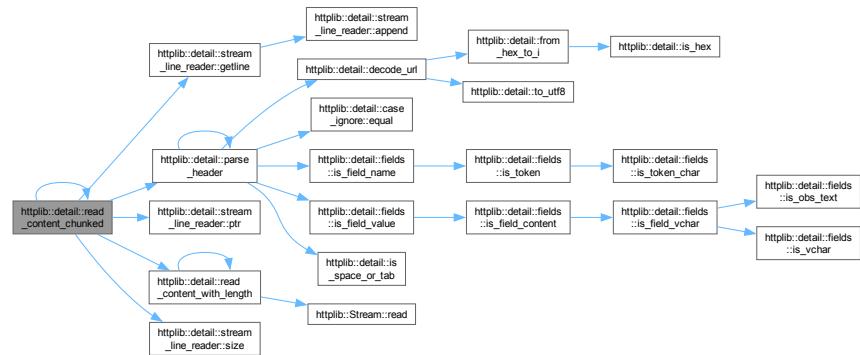
```

```
Stream & strm,
T & x,
ContentReceiverWithProgress out) [inline]
```

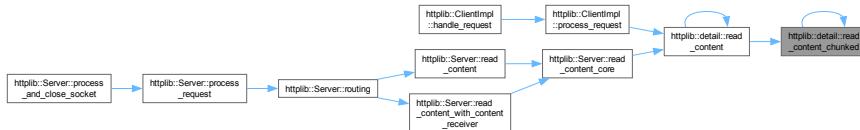
См. определение в файле `httpplib.h` строка 4412

```
04413                                     {
04414     const auto bufsiz = 16;
04415     char buf[bufsiz];
04416
04417     stream_line_reader line_reader(strm, buf, bufsiz);
04418
04419     if (!line_reader.getline()) { return false; }
04420
04421     unsigned long chunk_len;
04422     while (true) {
04423         char *end_ptr;
04424
04425         chunk_len = std::strtoul(line_reader.ptr(), &end_ptr, 16);
04426
04427         if (end_ptr == line_reader.ptr()) { return false; }
04428         if (chunk_len == ULONG_MAX) { return false; }
04429
04430         if (chunk_len == 0) { break; }
04431
04432         if (!read_content_with_length(strm, chunk_len, nullptr, out)) {
04433             return false;
04434         }
04435
04436         if (!line_reader.getline()) { return false; }
04437
04438         if (strcmp(line_reader.ptr(), "\r\n") != 0) { return false; }
04439
04440         if (!line_reader.getline()) { return false; }
04441     }
04442
04443     assert(chunk_len == 0);
04444
04445 // NOTE: In RFC 9112, '7.1 Chunked Transfer Coding' mentions "The chunked
04446 // transfer coding is complete when a chunk with a chunk-size of zero is
04447 // received, possibly followed by a trailer section, and finally terminated by
04448 // an empty line". https://www.rfc-editor.org/rfc/rfc9112.html#section-7.1
04449 //
04450 // In '7.1.3. Decoding Chunked', however, the pseudo-code in the section
04451 // does't care for the existence of the final CRLF. In other words, it seems
04452 // to be ok whether the final CRLF exists or not in the chunked data.
04453 // https://www.rfc-editor.org/rfc/rfc9112.html#section-7.1.3
04454 //
04455 // According to the reference code in RFC 9112, cpp-httplib now allows
04456 // chunked transfer coding data without the final CRLF.
04457 if (!line_reader.getline()) { return true; }
04458
04459 while (strcmp(line_reader.ptr(), "\r\n") != 0) {
04460     if (line_reader.size() > CPPHTTPPLIB_HEADER_MAX_LENGTH) { return false; }
04461
04462 // Exclude line terminator
04463 constexpr auto line_terminator_len = 2;
04464 auto end = line_reader.ptr() + line_reader.size() - line_terminator_len;
04465
04466 parse_header(line_reader.ptr(), end,
04467     [&](const std::string &key, const std::string &val) {
04468         x.headers.emplace(key, val);
04469     });
04470
04471     if (!line_reader.getline()) { return false; }
04472 }
04473
04474 return true;
04475 }
```

Граф вызовов:



Граф вызова функции:



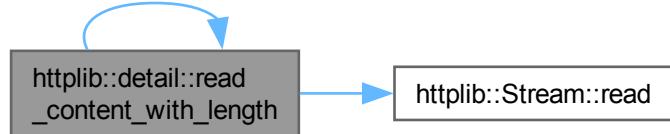
5.3.2.65 `read_content_with_length()`

```
bool httpplib::detail::read_content_with_length (
    Stream & strm,
    uint64_t len,
    Progress progress,
    ContentReceiverWithProgress out) [inline]
```

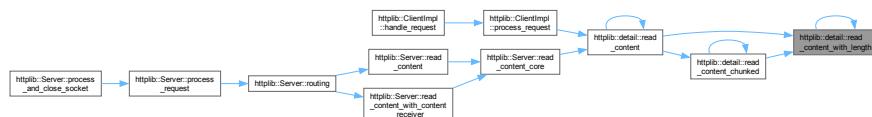
См. определение в файле `httpplib.h` строка 4362

```
04364
04365     char buf[CPPHTTPPLIB_RECV_BUFSIZ];
04366
04367     uint64_t r = 0;
04368     while (r < len) {
04369         auto read_len = static_cast<size_t>(len - r);
04370         auto n = strm.read(buf, std::min)(read_len, CPPHTTPPLIB_RECV_BUFSIZ));
04371         if (n <= 0) { return false; }
04372
04373         if (!out(buf, static_cast<size_t>(n), r, len)) { return false; }
04374         r += static_cast<uint64_t>(n);
04375
04376         if (progress) {
04377             if (!progress(r, len)) { return false; }
04378         }
04379     }
04380
04381     return true;
04382 }
```

Граф вызовов:



Граф вызова функции:



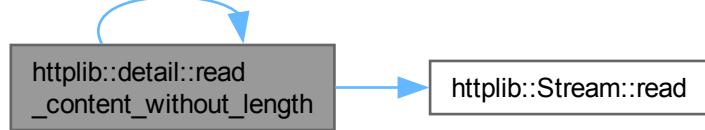
5.3.2.66 `read_content_without_length()`

```
bool httpplib::detail::read_content_without_length (
    Stream & strm,
    ContentReceiverWithProgress out) [inline]
```

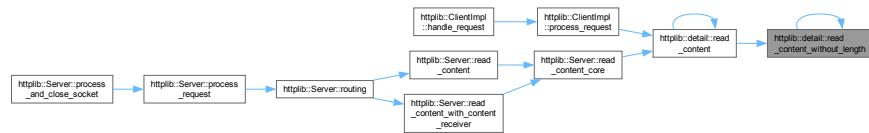
См. определение в файле `httpplib.h` строка 4395

```
04396     {
04397     char buf[CPPHTTPPLIB_RECV_BUFSIZ];
04398     uint64_t r = 0;
04399     for (;;) {
04400         auto n = strm.read(buf, CPPHTTPPLIB_RECV_BUFSIZ);
04401         if (n == 0) { return true; }
04402         if (n < 0) { return false; }
04403         if (!out(buf, static_cast<size_t>(n), r, 0)) { return false; }
04404         r += static_cast<uint64_t>(n);
04405     }
04406     return true;
04407 }
04408 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.67 `read_headers()`

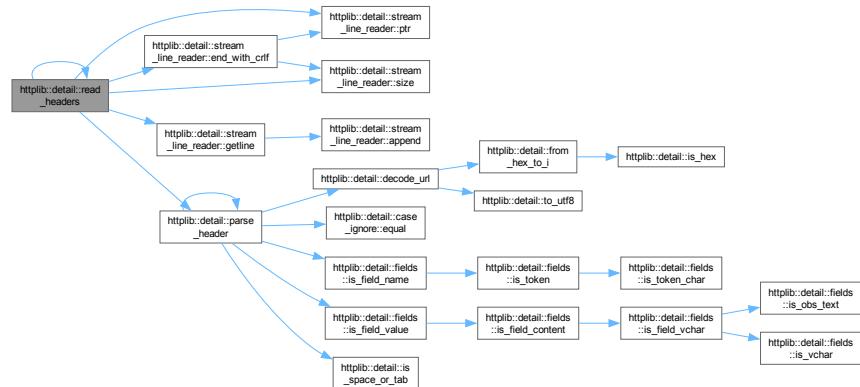
```
bool httpplib::detail::read_headers (
    Stream & strm,
    Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 4323

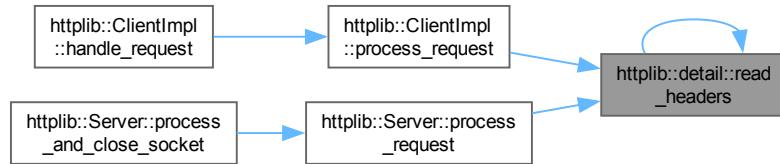
```

04323 {
04324     const auto bufsiz = 2048;
04325     char buf[bufsiz];
04326     stream_line_reader line_reader(strm, buf, bufsiz);
04327
04328     for (;;) {
04329         if (!line_reader.getline()) { return false; }
04330
04331         // Check if the line ends with CRLF.
04332         auto line_terminator_len = 2;
04333         if (line_reader.end_with_crlf()) {
04334             // Blank line indicates end of headers.
04335             if (line_reader.size() == 2) { break; }
04336         } else {
04337 #ifdef CPPHTTPPLIB_ALLOW_LF_AS_LINE_TERMINATOR
04338             // Blank line indicates end of headers.
04339             if (line_reader.size() == 1) { break; }
04340             line_terminator_len = 1;
04341 #else
04342             continue; // Skip invalid line.
04343 #endif
04344     }
04345     if (line_reader.size() > CPPHTTPPLIB_HEADER_MAX_LENGTH) { return false; }
04346
04347     // Exclude line terminator
04348     auto end = line_reader.ptr() + line_reader.size() - line_terminator_len;
04349
04350     if (!parse_header(line_reader.ptr(), end,
04351                         [&](const std::string &key, const std::string &val) {
04352                             headers.emplace(key, val);
04353                         })) {
04354         return false;
04355     }
04356 }
04357 }
04358
04359 return true;
04360 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.68 `read_socket()`

```
ssize_t httpplib::detail::read_socket (
    socket_t sock,
    void * ptr,
    size_t size,
    int flags) [inline]
```

См. определение в файле `httpplib.h` строка 3257

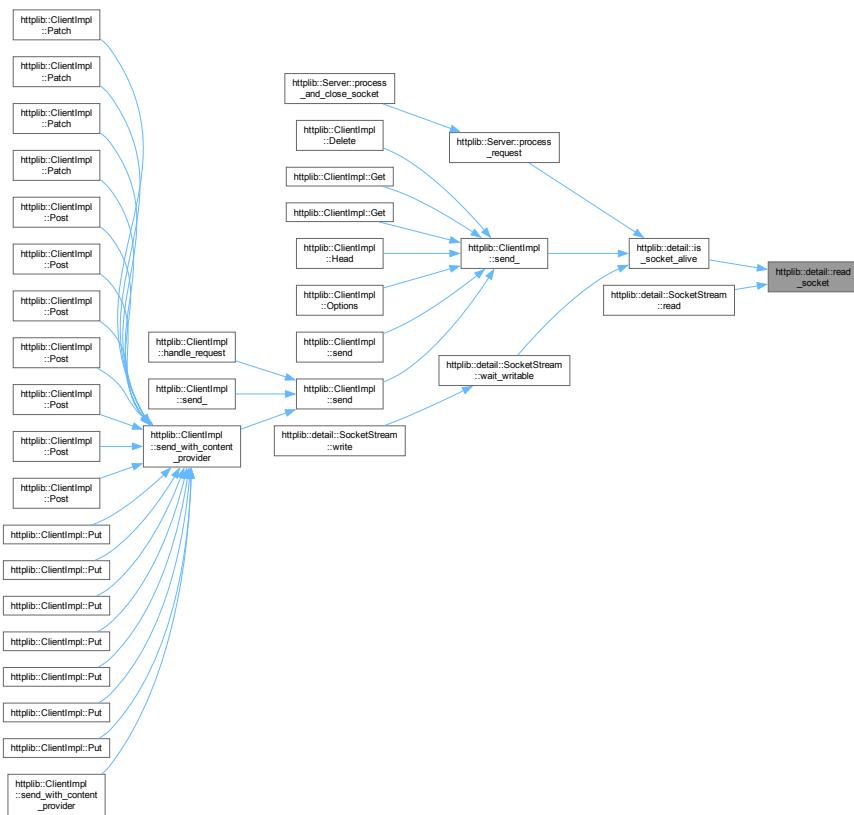
```

03257
03258     return handle_EINTR([&]() {
03259         return recv(sock,
03260 #ifdef _WIN32
03261             static_cast<char*>(ptr), static_cast<int>(size),
03262 #else
03263             ptr, size,
03264 #endif
03265             flags);
03266     });
03267 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.69 redirect()

```

template<typename T>
bool httpplib::detail::redirect (
    T & cli,
    Request & req,
    Response & res,
    const std::string & path,
    const std::string & location,
    Error & error) [inline]
  
```

См. определение в файле `httpplib.h` строка 4813

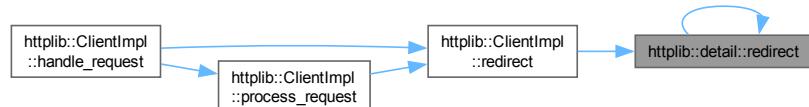
```

04815     {
04816     Request new_req = req;
04817     new_req.path = path;
04818     new_req.redirect_count_ -= 1;
04819
04820     if (res.status == StatusCode::SeeOther_303 &&
04821         (req.method != "GET" && req.method != "HEAD")) {
04822         new_req.method = "GET";
04823         new_req.body.clear();
04824         new_req.headers.clear();
04825     }
04826
04827     Response new_res;
04828
04829     auto ret = cli.send(new_req, new_res, error);
04830     if (ret) {
04831         req = new_req;
04832         res = new_res;
04833
04834         if (res.location.empty()) { res.location = location; }
04835     }
04836     return ret;
04837 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.70 select_impl()

```

template<bool Read>
ssize_t httpplib::detail::select_impl (
    socket_t sock,
    time_t sec,
    time_t usec) [inline]
```

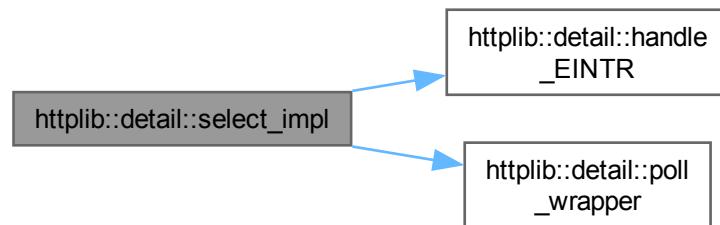
См. определение в файле `httpplib.h` строка 3291

```

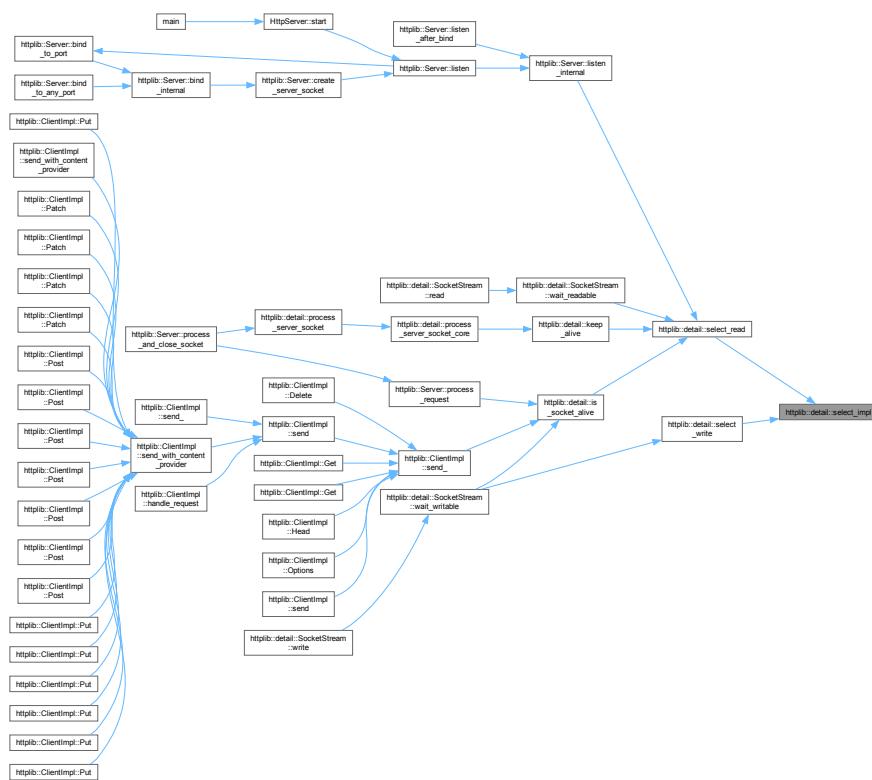
03291
03292     struct pollfd pfd;
03293     pfd.fd = sock;
03294     pfd.events = (Read ? POLLIN : POLLOUT);
```

```
03295
03296     auto timeout = static_cast<int>(sec * 1000 + usec / 1000);
03297
03298     return handle_EINTR([&]() { return poll_wrapper(&pfd, 1, timeout); });
03299 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.71 select_read()

```
ssize_t httplib::detail::select_read (
```

```
time_t sec,
time_t usec) [inline]
```

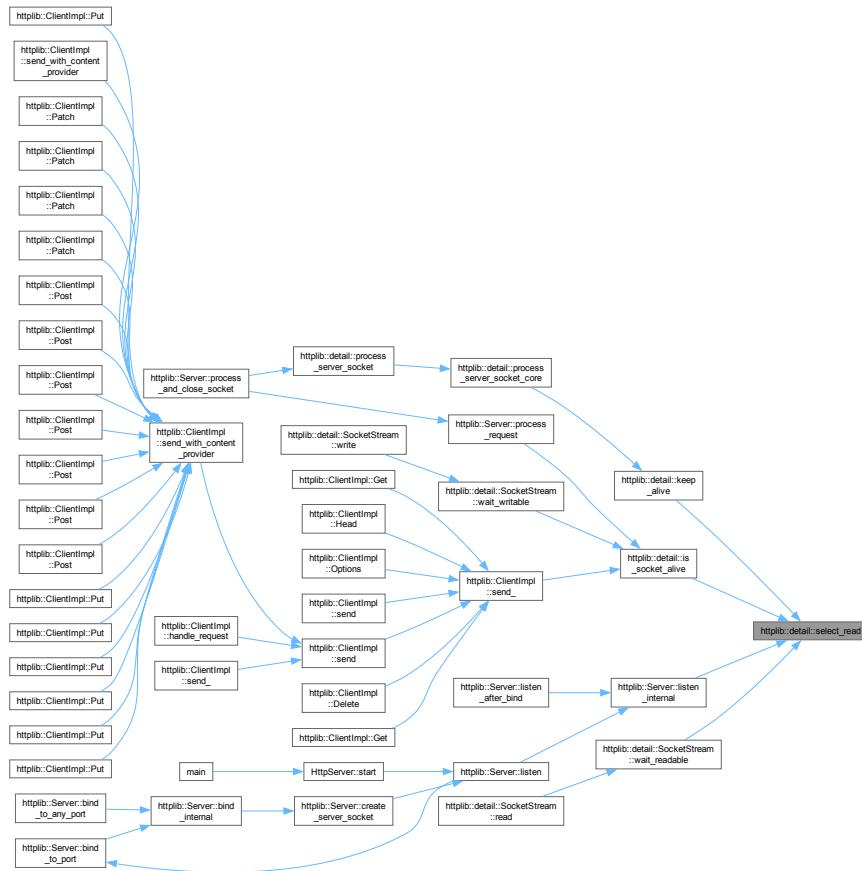
См. определение в файле [httpplib.h](#) строка 3301

```
03301 {  
03302     return select_impl<true>(sock, sec, usec);  
03303 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.72 `select_write()`

```
ssize_t httpplib::detail::select_write (
    socket_t sock,
    time_t sec,
    time_t usec) [inline]
```

См. определение в файле `httpplib.h` строка 3305

```
03305     {
03306     return select_impl<false>(sock, sec, usec);
03307 }
```

Граф вызовов:



Граф вызова функции:

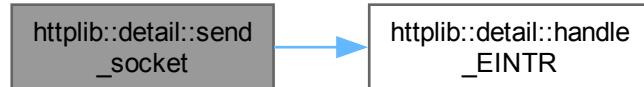
5.3.2.73 `send_socket()`

```
ssize_t httpplib::detail::send_socket (
    socket_t sock,
    const void * ptr,
    size_t size,
    int flags) [inline]
```

См. определение в файле `httpplib.h` строка 3269

```
03270     {
03271     return handle_EINTR([&]() {
03272         return send(sock,
03273 #ifdef _WIN32
03274             static_cast<const char *>(ptr), static_cast<int>(size),
03275 #else
03276             ptr, size,
03277 #endif
03278             flags);
03279 });
03280 }
```

Граф вызовов:



Граф вызова функции:



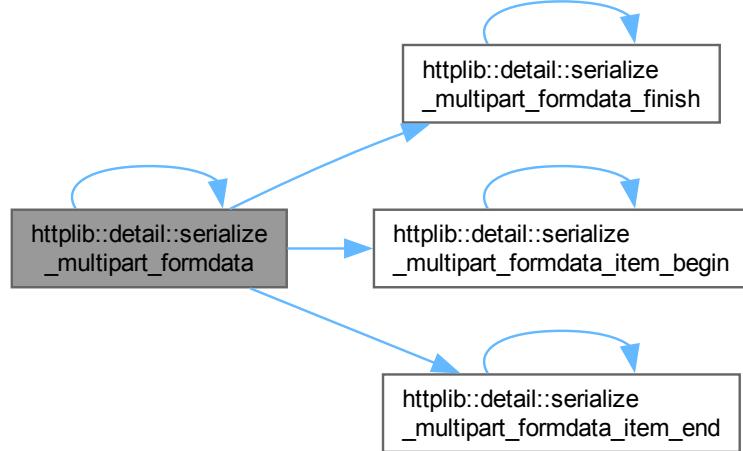
5.3.2.74 `serialize_multipart_formdata()`

```
std::string httpplib::detail::serialize_multipart_formdata (
    const MultipartFormDataItems & items,
    const std::string & boundary,
    bool finish = true) [inline]
```

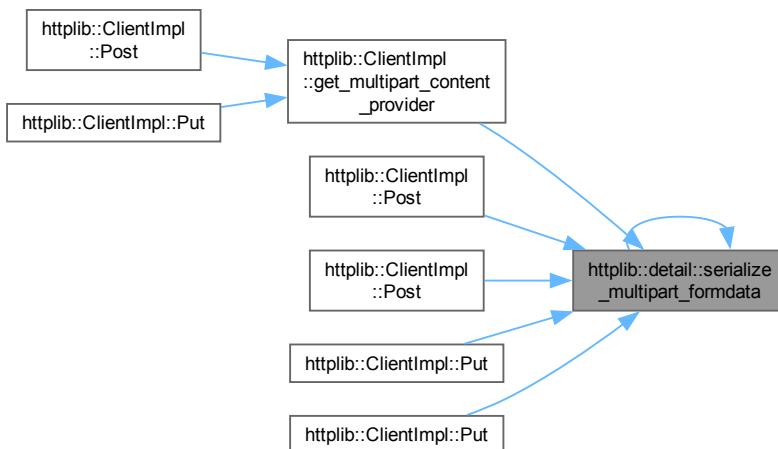
См. определение в файле [httpplib.h](#) строка 5277

```
05278     std::string body;
05279
05280     for (const auto &item : items) {
05281         body += serialize_multipart_formdata_item_begin(item, boundary);
05282         body += item.content + serialize_multipart_formdata_item_end();
05283     }
05284
05285     if (finish) { body += serialize_multipart_formdata_finish(boundary); }
05286
05287     return body;
05288 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.75 `serialize_multipart_formdata_finish()`

```
std::string httpplib::detail::serialize_multipart_formdata_finish (
    const std::string & boundary) [inline]
```

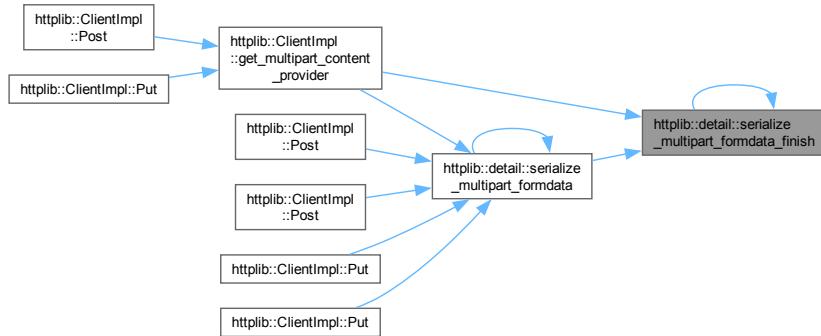
См. определение в файле `httpplib.h` строка 5267

```
05267
05268     return "--" + boundary + "--\r\n";
05269 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.76 `serialize_multipart_formdata_get_content_type()`

```
std::string httpplib::detail::serialize_multipart_formdata_get_content_type (
    const std::string & boundary) [inline]
```

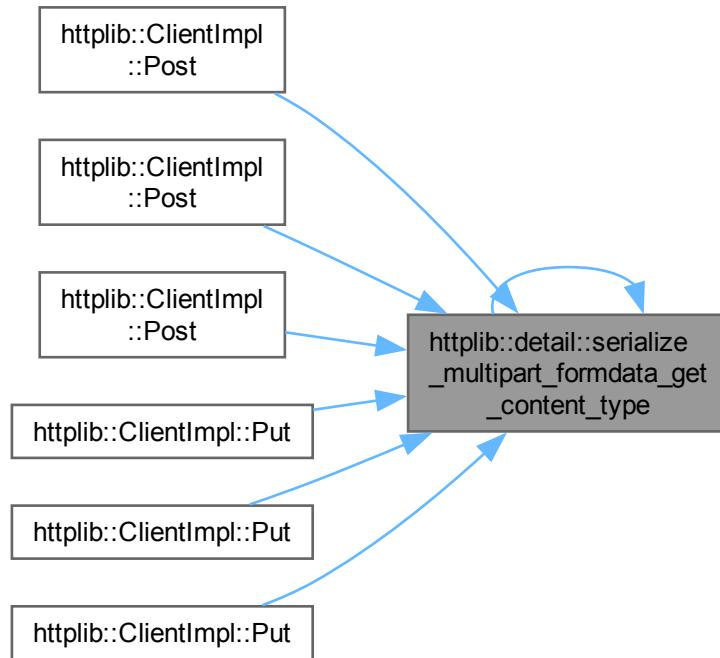
См. определение в файле [httpplib.h](#) строка 5272

```
05272 {
05273     return "multipart/form-data; boundary=" + boundary;
05274 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.77 `serialize_multipartFormData_item_begin()`

```
template<typename T>
std::string httplib::detail::serialize_multipartFormData_item_begin (
    const T & item,
    const std::string & boundary) [inline]
```

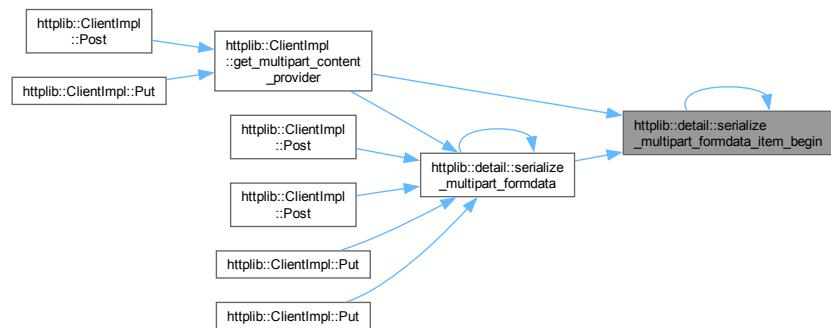
См. определение в файле [httplib.h](#) строка 5248

```
05249     {
05250     std::string body = "--" + boundary + "\r\n";
05251     body += "Content-Disposition: form-data; name=\"" + item.name + "\"";
05252     if (!item.filename.empty()) {
05253         body += "; filename=\"" + item.filename + "\"";
05254     }
05255     body += "\r\n";
05256     if (!item.content_type.empty()) {
05257         body += "Content-Type: " + item.content_type + "\r\n";
05258     }
05259     body += "\r\n";
05260
05261     return body;
05262 }
```

Граф вызовов:



Граф вызова функции:



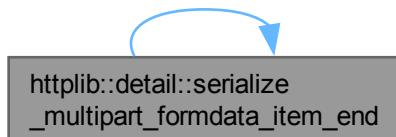
5.3.2.78 `serialize_multipart_formdata_item_end()`

```
std::string httpplib::detail::serialize_multipart_formdata_item_end () [inline]
```

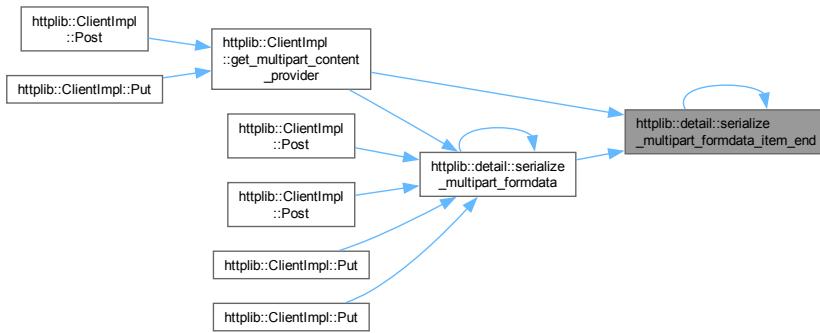
См. определение в файле [httpplib.h](#) строка [5264](#)

```
05264 { return "\r\n"; }
```

Граф вызовов:



Граф вызова функции:



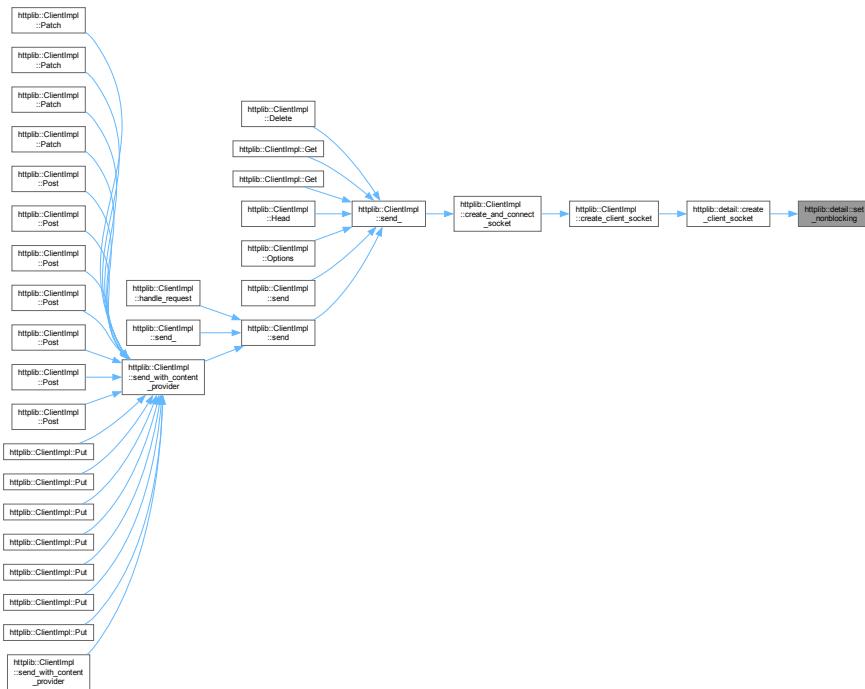
5.3.2.79 set_nonblocking()

```
void http://detail::set_nonblocking (
    socket_t sock,
    bool nonblocking) [inline]
```

См. определение в файле `http://http.h` строка 3661

```
03661
03662 #ifdef _WIN32
03663 auto flags = nonblocking ? 1UL : 0UL;
03664 ioctlsocket(sock, FIONBIO, &flags);
03665 #else
03666 auto flags = fcntl(sock, F_GETFL, 0);
03667 fcntl(sock, F_SETFL,
03668     nonblocking ? (flags | O_NONBLOCK) : (flags & (~O_NONBLOCK)));
03669 #endif
03670 }
```

Граф вызова функции:



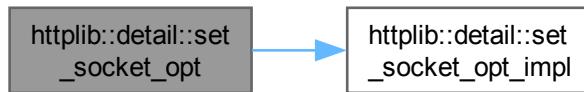
5.3.2.80 `set_socket_opt()`

```
bool httpplib::detail::set_socket_opt (
    socket_t sock,
    int level,
    int optname,
    int opt) [inline]
```

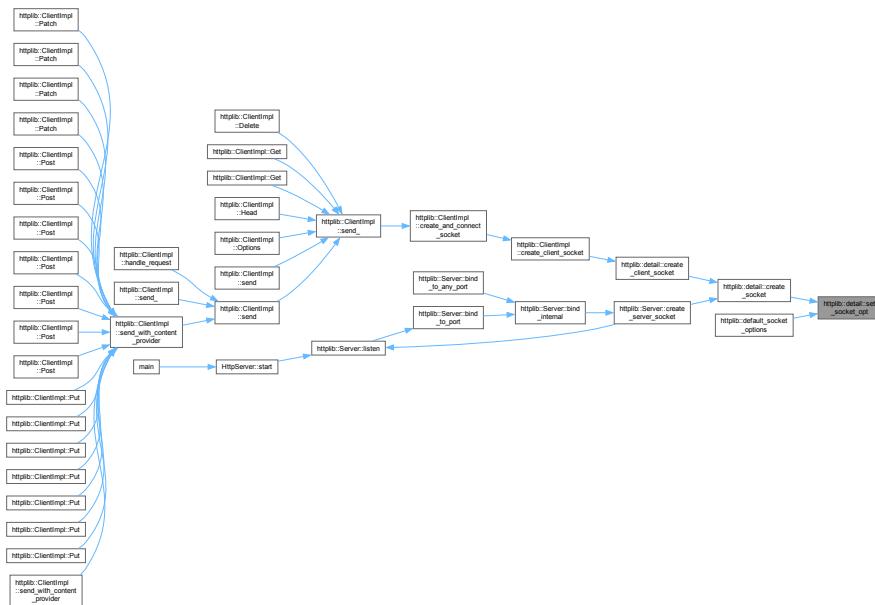
См. определение в файле `httpplib.h` строка 2117

```
02117 {  
02118     return set_socket_opt_impl(sock, level, optname, &optval, sizeof(optval));  
02119 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.81 `set_socket_opt_Impl()`

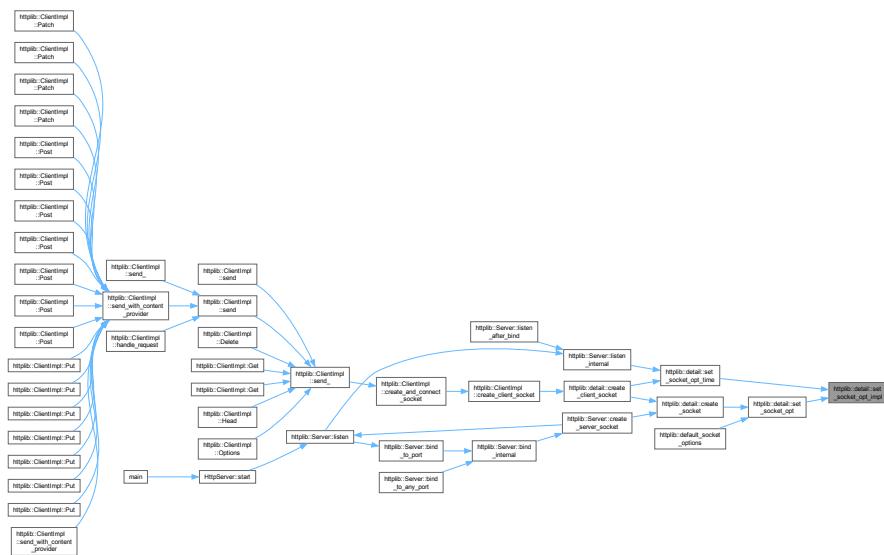
```
bool httpplib::detail::set_socket_opt_Impl (
    socket_t sock,
```

```
int level,
int optname,
const void * optval,
socklen_t optlen) [inline]
```

См. определение в файле `httpplib.h` строка 2106

```
02107 {  
02108     return setsockopt(sock, level, optname,  
02109 #ifdef _WIN32  
02110         reinterpret_cast<const char *>(optval),  
02111 #else  
02112         optval,  
02113 #endif  
02114         optlen) == 0;  
02115 }
```

Граф вызова функции:



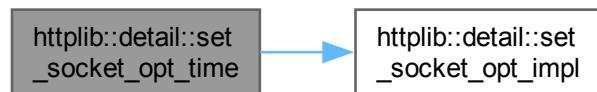
5.3.2.82 `set_socket_opt_time()`

```
bool httpplib::detail::set_socket_opt_time (
    socket_t sock,
    int level,
    int optname,
    time_t sec,
    time_t usec) [inline]
```

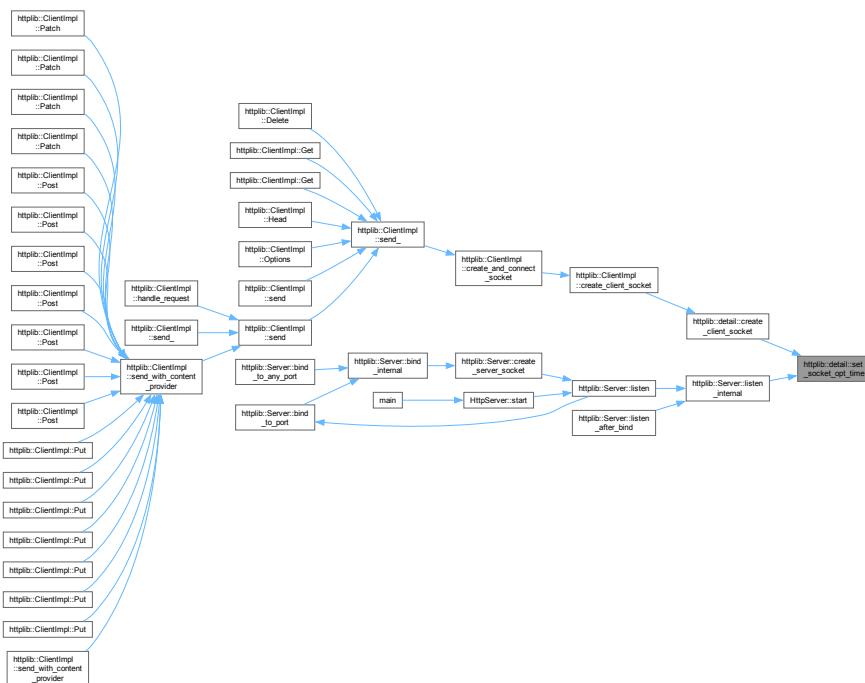
См. определение в файле `httpplib.h` строка 2121

```
02122 {  
02123 #ifdef _WIN32  
02124     auto timeout = static_cast<uint32_t>(sec * 1000 + usec / 1000);  
02125 #else  
02126     timeval timeout;  
02127     timeout.tv_sec = static_cast<long>(sec);  
02128     timeout.tv_usec = static_cast<decltype(timeout.tv_usec)>(usec);  
02129 #endif  
02130     return set_socket_opt_impl(sock, level, optname, &timeout, sizeof(timeout));  
02131 }
```

Граф вызовов:



Граф вызова функции:



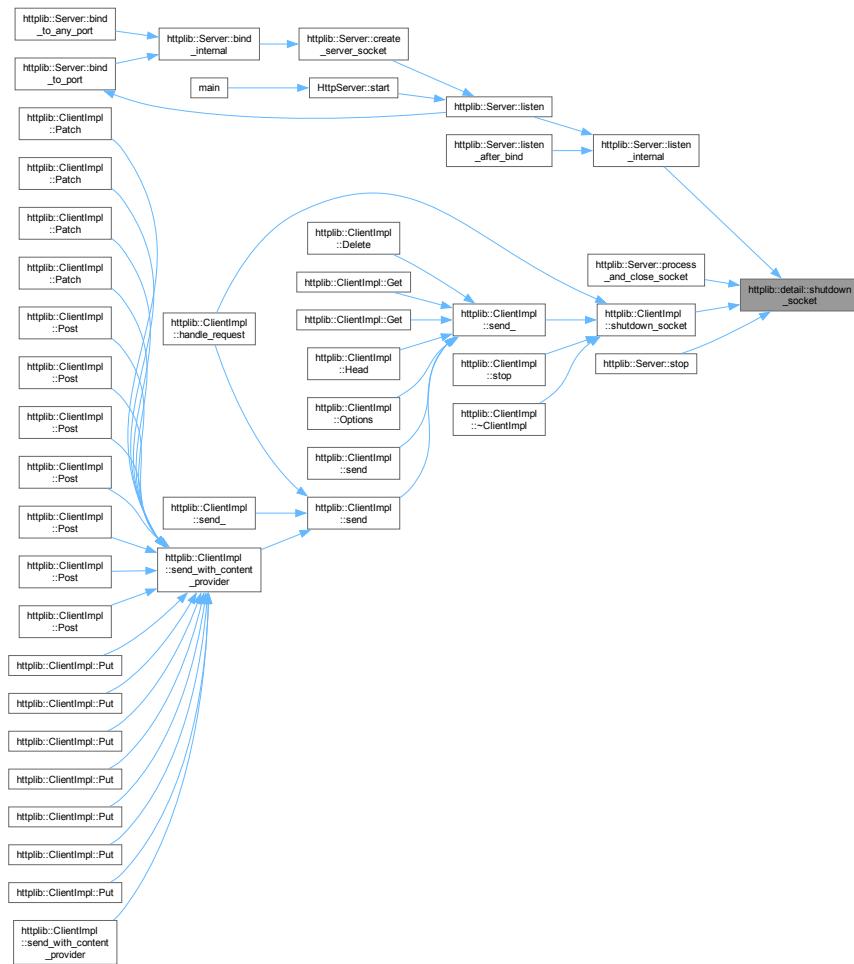
5.3.2.83 shutdown_socket()

```
int httplib::detail::shutdown_socket (
```

См. определение в файле `httpplib.h` строка 3492

```
03492 {  
03493 #ifndef _WIN32  
03494     return shutdown(sock, SD_BOTH);  
03495 #else  
03496     return shutdown(sock, SHUT_RDWR);  
03497 #endif  
03498 }
```

Граф вызова функции:



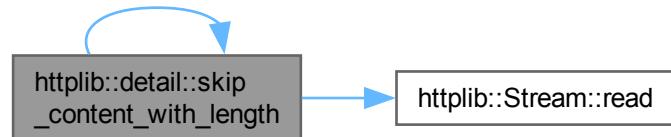
5.3.2.84 skip content with length()

```
void httplib::detail::skip_content_with_length (
    Stream & strm,
    uint64 t len) [inline]
```

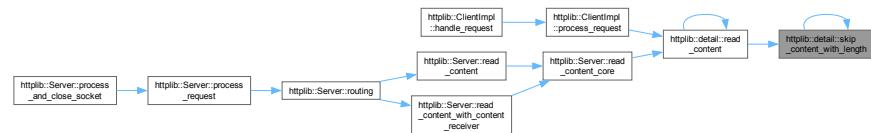
См. определение в файле [httplib.h](#) строка 4384

```
04384     {  
04385     char buf[CPPHTTPLIB_RECV_BUFSIZ];  
04386     uint64_t r = 0;  
04387     while (r < len) {  
04388         auto read_len = static_cast<size_t>(len - r);  
04389         auto n = strm.read(buf, std::min)(read_len, CPPHTTPLIB_RECV_BUFSIZ);  
04390         if (n <= 0) { return; }  
04391         r += static_cast<uint64_t>(n);  
04392     }  
04393 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.85 `split()` [1/2]

```

void httpplib::detail::split (
    const char * b,
    const char * e,
    char d,
    size_t m,
    std::function< void(const char *, const char *)> fn) [inline]
  
```

См. определение в файле [httpplib.h](#) строка 3013

```

03014     size_t i = 0;
03015     size_t beg = 0;
03016     size_t count = 1;
03017
03018     while (e ? (b + i < e) : (b[i] != '\0')) {
03019         if (b[i] == d && count < m) {
03020             auto r = trim(b, e, beg, i);
03021             if (r.first < r.second) { fn(&b[r.first], &b[r.second]); }
03022             beg = i + 1;
03023             count++;
03024         }
03025     }
03026     i++;
03027 }
03028     if (i) {
03029         auto r = trim(b, e, beg, i);
03030         if (r.first < r.second) { fn(&b[r.first], &b[r.second]); }
03031     }
03032 }
03033 }
  
```

Граф вызовов:



5.3.2.86 `split()` [2/2]

```

void httpplib::detail::split (
    const char * b,
    const char * e,
    char d,
    std::function< void(const char *, const char *)> fn) [inline]
  
```

См. определение в файле [httpplib.h](#) строка 3008

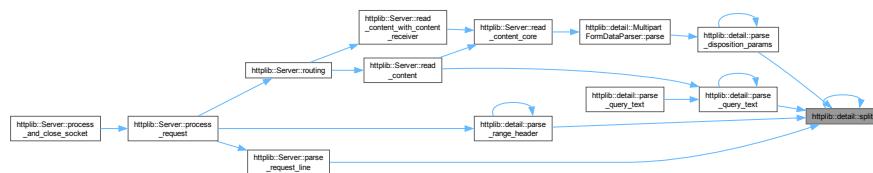
```

03009 {
03010     return split(b, e, d, (std::numeric_limits<size_t>::max)(), std::move(fn));
03011 }
  
```

Граф вызовов:



Граф вызова функции:



5.3.2.87 `str2tag()`

```
unsigned int httpplib::detail::str2tag (
    const std::string & s) [inline]
```

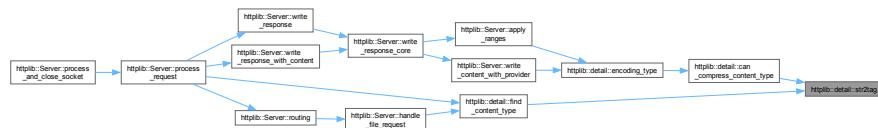
См. определение в файле `httpplib.h` строка 3876

```
03876     {
03877     return str2tag_core(s.data(), s.size(), 0);
03878 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.88 `str2tag_core()`

```
unsigned int httpplib::detail::str2tag_core (
    const char * s,
    size_t l,
    unsigned int h) [inline], [constexpr]
```

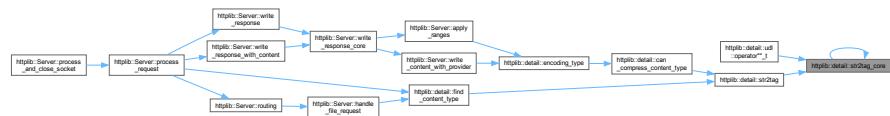
См. определение в файле `httpplib.h` строка 3864

```
03865     {
03866     return (l == 0)
03867     ? h
03868     : str2tag_core(
03869         s + 1, l - 1,
03870         // Unsets the 6 high bits of h, therefore no overflow happens
03871         ((std::numeric_limits<unsigned int>::max)() >> 6) &
03872         h * 33) ^
03873         static_cast<unsigned char>(*s));
03874 }
```

Граф вызовов:



Граф вызова функции:



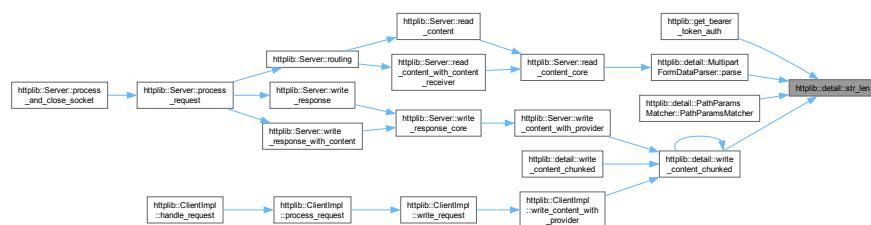
5.3.2.89 str_len()

```
template<size_t N>
size_t httpplib::detail::str_len (
    const char(&)[N]) [inline], [constexpr]
```

См. определение в файле [httpplib.h](#) строка 2060

```
02060 {  
02061     return N - 1;  
02062 }
```

Граф вызова функции:



5.3.2.90 to_utf8()

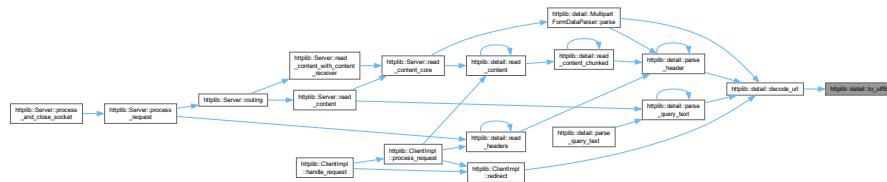
```
size_t httpplib::detail::to_utf8 (
    int code,
    char * buff) [inline]
```

См. определение в файле `httpplib.h` строка 2746

```

02746     if (code < 0x0080) {
02747         buff[0] = static_cast<char>(code & 0x7F);
02748         return 1;
02749     } else if (code < 0x0800) {
02750         buff[0] = static_cast<char>(0xC0 | ((code >> 6) & 0x1F));
02751         buff[1] = static_cast<char>(0x80 | (code & 0x3F));
02752         return 2;
02753     } else if (code < 0xD800) {
02754         buff[0] = static_cast<char>(0xE0 | ((code >> 12) & 0xF));
02755         buff[1] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02756         buff[2] = static_cast<char>(0x80 | (code & 0x3F));
02757         return 3;
02758     } else if (code < 0xE000) { // D800 - DFFF is invalid...
02759         return 0;
02760     } else if (code < 0x10000) {
02761         buff[0] = static_cast<char>(0xE0 | ((code >> 12) & 0xF));
02762         buff[1] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02763         buff[2] = static_cast<char>(0x80 | (code & 0x3F));
02764         return 3;
02765     } else if (code < 0x110000) {
02766         buff[0] = static_cast<char>(0xF0 | ((code >> 18) & 0x7));
02767         buff[1] = static_cast<char>(0x80 | ((code >> 12) & 0x3F));
02768         buff[2] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02769         buff[3] = static_cast<char>(0x80 | (code & 0x3F));
02770         return 4;
02771     }
02772 }
02773 // NOTREACHED
02774 return 0;
02775 }
```

Граф вызова функции:



5.3.2.91 trim()

```

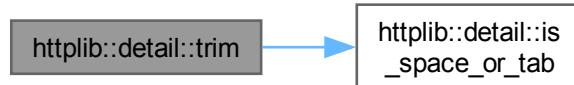
std::pair< size_t, size_t > httpplib::detail::trim (
    const char * b,
    const char * e,
    size_t left,
    size_t right) [inline]
```

См. определение в файле `httpplib.h` строка 2964

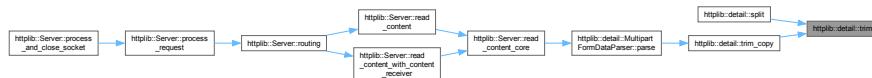
```

02965     {
02966     while (b + left < e && is_space_or_tab(b[left])) {
02967         left++;
02968     }
02969     while (right > 0 && is_space_or_tab(b[right - 1])) {
02970         right--;
02971     }
02972     return std::make_pair(left, right);
02973 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.92 `trim_copy()`

```
std::string httpplib::detail::trim_copy(
    const std::string & s) [inline]
```

См. определение в файле `httpplib.h` строка 2975

```
02975     {
02976     auto r = trim(s.data(), s.data() + s.size(), 0, s.size());
02977     return s.substr(r.first, r.second - r.first);
02978 }
```

Граф вызовов:



Граф вызова функции:



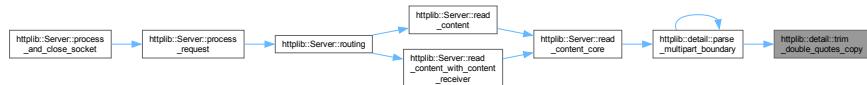
5.3.2.93 `trim_double_quotes_copy()`

```
std::string httpplib::detail::trim_double_quotes_copy (
    const std::string & s) [inline]
```

См. определение в файле [httpplib.h](#) строка 2980

```
02980     if (s.length() >= 2 && s.front() == '\"' && s.back() == '\"') {
02981         return s.substr(1, s.size() - 2);
02982     }
02983 }
02984 return s;
02985 }
```

Граф вызова функции:



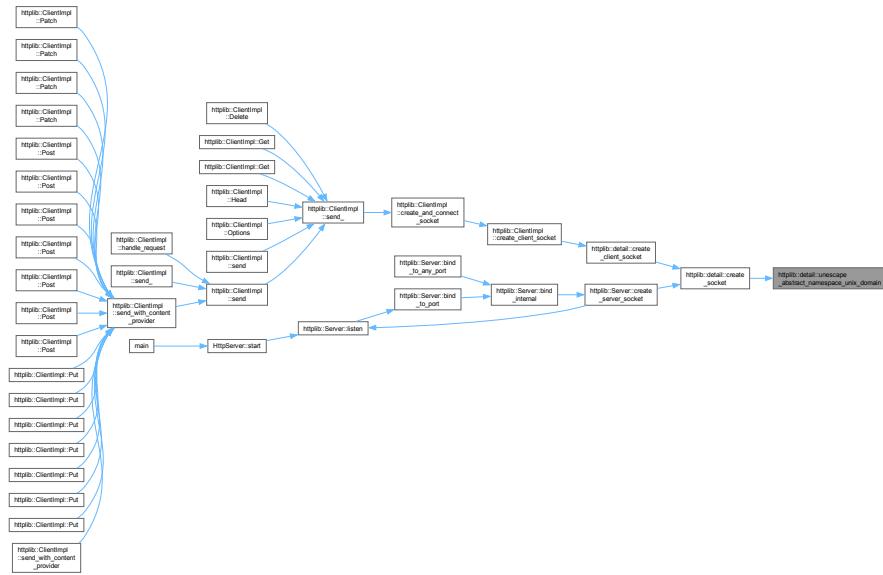
5.3.2.94 `unescape_abstract_namespace_unix_domain()`

```
std::string httpplib::detail::unescape_abstract_namespace_unix_domain (
    const std::string & s) [inline]
```

См. определение в файле [httpplib.h](#) строка 3510

```
03510 {
03511     if (s.size() > 1 && s[0] == '@') {
03512         auto ret = s;
03513         ret[0] = '\0';
03514         return ret;
03515     }
03516     return s;
03517 }
```

Граф вызова функции:



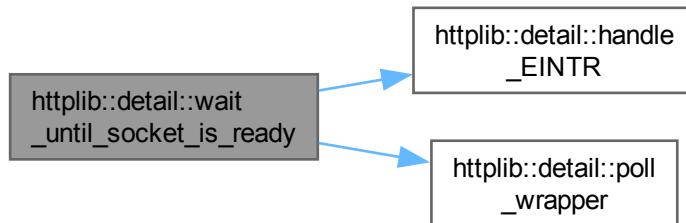
5.3.2.95 `wait_until_socket_is_ready()`

```
Error httpplib::detail::wait_until_socket_is_ready (
    socket_t sock,
    time_t sec,
    time_t usec) [inline]
```

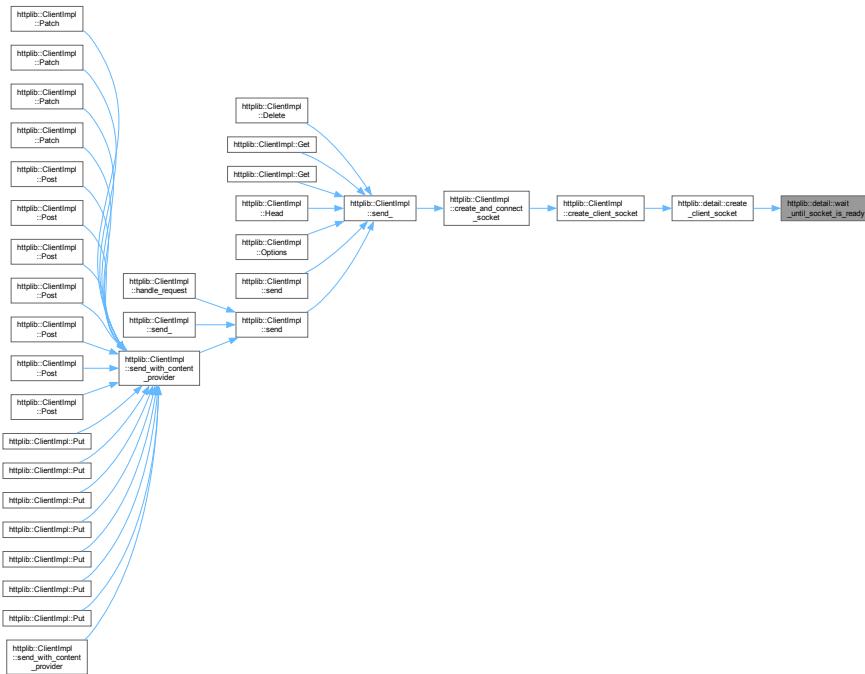
См. определение в файле `httpplib.h` строка 3309

```
03310                                     {
03311     struct pollfd pfd_read;
03312     pfd_read.fd = sock;
03313     pfd_read.events = POLLIN | POLLOUT;
03314
03315     auto timeout = static_cast<int>(sec * 1000 + usec / 1000);
03316
03317     auto poll_res =
03318         handle_EINTR([&]() { return poll_wrapper(&pfd_read, 1, timeout); });
03319
03320     if (poll_res == 0) { return Error::ConnectionTimeout; }
03321
03322     if (poll_res > 0 && pfd_read.revents & (POLLIN | POLLOUT)) {
03323         auto error = 0;
03324         socklen_t len = sizeof(error);
03325         auto res = getsockopt(sock, SOL_SOCKET, SO_ERROR,
03326             reinterpret_cast<char*>(&error), &len);
03327         auto successful = res >= 0 && !error;
03328         return successful ? Error::Success : Error::Connection;
03329     }
03330
03331     return Error::Connection;
03332 }
```

Граф вызовов:



Граф вызова функции:



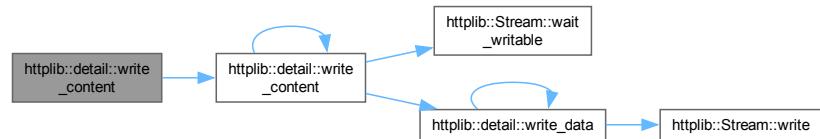
5.3.2.96 write_content() [1/2]

```
template<typename T>
bool httplib::detail::write_content (
    Stream & strm,
    const ContentProvider & content_provider,
    size_t offset,
    size_t length,
    const T & is_shutting_down) [inline]
```

См. определение в файле [httplib.h](#) строка 4662

```
04664                                         {  
04665     auto error = Error::Success;  
04666     return write_content(strm, content_provider, offset, length, is_shutting_down,  
04667                     error);  
04668 }
```

Граф вызовов:



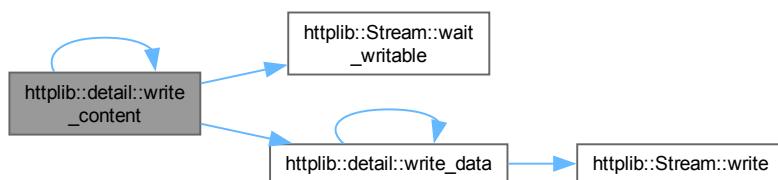
5.3.2.97 `write_content()` [2/2]

```
template<typename T>
bool httpplib::detail::write_content (
    Stream & strm,
    const ContentProvider & content_provider,
    size_t offset,
    size_t length,
    T is_shutting_down,
    Error & error) [inline]
```

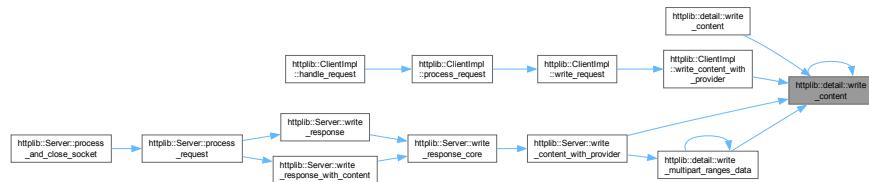
См. определение в файле [httpplib.h](#) строка 4624

```
04626     {
04627     size_t end_offset = offset + length;
04628     auto ok = true;
04629     DataSink data_sink;
04630
04631     data_sink.write = [&](const char *d, size_t l) -> bool {
04632         if (ok) {
04633             if (write_data(strm, d, l)) {
04634                 offset += l;
04635             } else {
04636                 ok = false;
04637             }
04638         }
04639         return ok;
04640     };
04641
04642     data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04643
04644     while (offset < end_offset && !is_shutting_down()) {
04645         if (!strm.wait_writable()) {
04646             error = Error::Write;
04647             return false;
04648         } else if (!content_provider(offset, end_offset - offset, data_sink)) {
04649             error = Error::Canceled;
04650             return false;
04651         } else if (!ok) {
04652             error = Error::Write;
04653             return false;
04654         }
04655     }
04656
04657     error = Error::Success;
04658     return true;
04659 }
```

Граф вызовов:



Граф вызова функции:



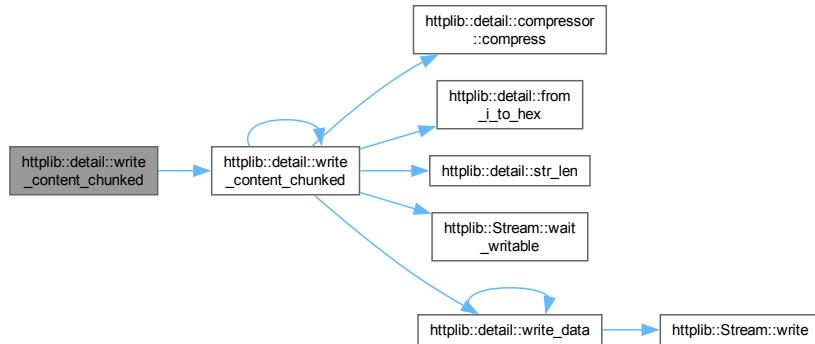
5.3.2.98 `write_content_chunked()` [1/2]

```
template<typename T, typename U>
bool httpplib::detail::write_content_chunked (
    Stream & strm,
    const ContentProvider & content_provider,
    const T & is_shutting_down,
    U & compressor) [inline]
```

См. определение в файле `httpplib.h` строка 4804

```
04806
04807     auto error = Error::Success;
04808     return write_content_chunked(strm, content_provider, is_shutting_down,
04809                                         compressor, error);
04810 }
```

Граф вызовов:



5.3.2.99 `write_content_chunked()` [2/2]

```
template<typename T, typename U>
bool httpplib::detail::write_content_chunked (
    Stream & strm,
    const ContentProvider & content_provider,
    const T & is_shutting_down,
```

```
U & compressor,
Error & error) [inline]
```

См. определение в файле [httpplib.h](#) строка 4706

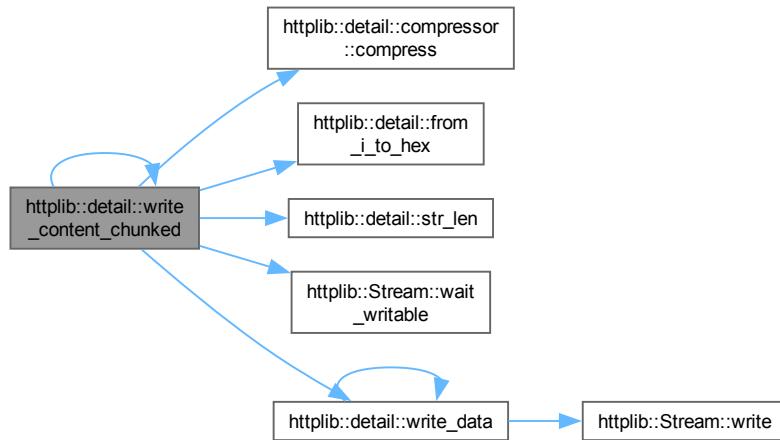
```
04707 size_t offset = 0;
04708 auto data_available = true;
04709 auto ok = true;
04710 DataSink data_sink;
04711
04712 data_sink.write = [&](const char *d, size_t l) -> bool {
04713     if (ok) {
04714         data_available = l > 0;
04715         offset += l;
04716
04717         std::string payload;
04718         if (compressor.compress(d, l, false,
04719             [&](const char *data, size_t data_len) {
04720                 payload.append(data, data_len);
04721                 return true;
04722             })) {
04723             if (!payload.empty()) {
04724                 // Emit chunked response header and footer for each chunk
04725                 auto chunk =
04726                     from_i_to_hex(payload.size()) + "\r\n" + payload + "\r\n";
04727                 if (!write_data(strm, chunk.data(), chunk.size())) { ok = false; }
04728             }
04729         } else {
04730             ok = false;
04731         }
04732     }
04733     return ok;
04734 };
04735
04736 data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04737
04738 auto done_with_trailer = [&](const Headers *trailer) {
04739     if (!ok) { return; }
04740
04741     data_available = false;
04742
04743     std::string payload;
04744     if (!compressor.compress(nullptr, 0, true,
04745             [&](const char *data, size_t data_len) {
04746                 payload.append(data, data_len);
04747                 return true;
04748             })) {
04749         ok = false;
04750         return;
04751     }
04752
04753     if (!payload.empty()) {
04754         // Emit chunked response header and footer for each chunk
04755         auto chunk = from_i_to_hex(payload.size()) + "\r\n" + payload + "\r\n";
04756         if (!write_data(strm, chunk.data(), chunk.size())) {
04757             ok = false;
04758             return;
04759         }
04760     }
04761 }
04762
04763 constexpr const char done_marker[] = "0\r\n";
04764 if (!write_data(strm, done_marker, str_len(done_marker))) { ok = false; }
04765
04766 // Trailer
04767 if (trailer) {
04768     for (const auto &kv : *trailer) {
04769         std::string field_line = kv.first + ": " + kv.second + "\r\n";
04770         if (!write_data(strm, field_line.data(), field_line.size())) {
04771             ok = false;
04772         }
04773     }
04774 }
04775
04776 constexpr const char crlf[] = "\r\n";
04777 if (!write_data(strm, crlf, str_len(crlf))) { ok = false; }
04778 }
04779
04780 data_sink.done = [&](void) { done_with_trailer(nullptr); };
04781
04782 data_sink.done_with_trailer = [&](const Headers &trailer) {
04783     done_with_trailer(&trailer);
04784 };
04785
04786 while (data_available && !is_shutting_down()) {
04787     if (!strm.wait_writable()) {
```

```

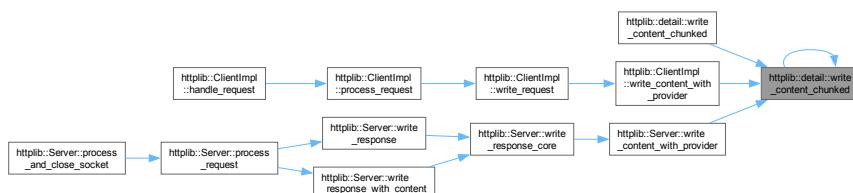
04788     error = Error::Write;
04789     return false;
04790 } else if (!content_provider(offset, 0, data_sink)) {
04791     error = Error::Canceled;
04792     return false;
04793 } else if (!ok) {
04794     error = Error::Write;
04795     return false;
04796 }
04797 }
04798
04799 error = Error::Success;
04800 return true;
04801 }

```

Граф вызовов:



Граф вызова функции:



5.3.2.100 write_content_without_length()

```

template<typename T>
bool http://lib::detail::write_content_without_length (
    Stream & strm,
    const ContentProvider & content_provider,
    const T & is_shutting_down) [inline]

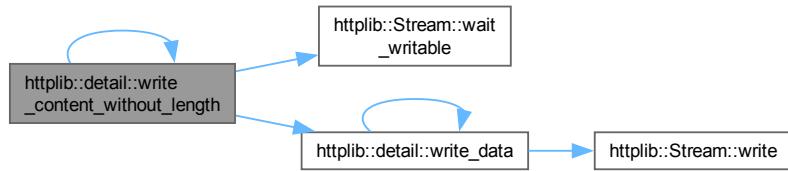
```

См. определение в файле [http://lib.h](#) строка 4672

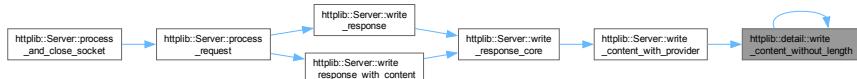
```

04674     size_t offset = 0;
04675     auto data_available = true;
04676     auto ok = true;
04677     DataSink data_sink;
04678
04679     data_sink.write = [&](const char *d, size_t l) -> bool {
04680         if (ok) {
04681             offset += l;
04682             if (!write_data(strm, d, l)) { ok = false; }
04683         }
04684         return ok;
04685     };
04686
04687     data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04688
04689     data_sink.done = [&](void) { data_available = false; };
04690
04691     while (data_available && !is_shutting_down()) {
04692         if (!strm.wait_writable()) {
04693             return false;
04694         } else if (!content_provider(offset, 0, data_sink)) {
04695             return false;
04696         } else if (!ok) {
04697             return false;
04698         }
04699     }
04700
04701     return true;
04702 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.101 `write_data()`

```

bool httpplib::detail::write_data (
    Stream & strm,
    const char * d,
    size_t l) [inline]
```

См. определение в файле `httpplib.h` строка 4613

```

04613
04614     size_t offset = 0;
04615     while (offset < l) {
04616         auto length = strm.write(d + offset, l - offset);
04617         if (length < 0) { return false; }
```

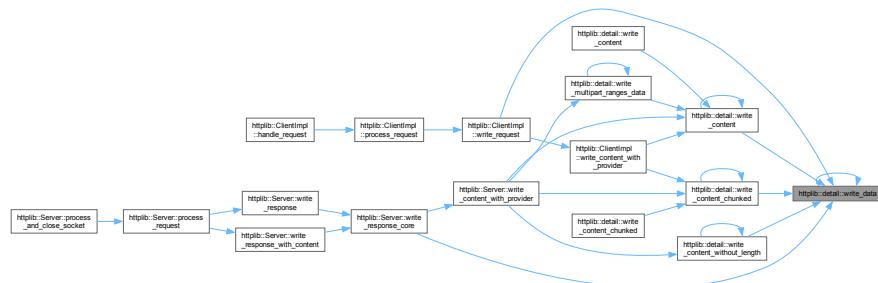
```

04618     offset += static_cast<size_t>(length);
04619 }
04620 return true;
04621 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.102 `write_headers()`

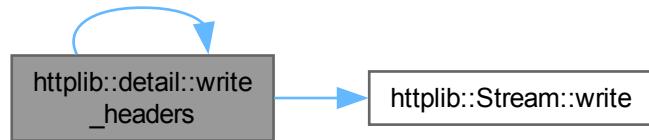
```
ssize_t httpplib::detail::write_headers (
    Stream & strm,
    const Headers & headers) [inline]
```

См. определение в файле [httpplib.h](#) строка 4594

```

04594 {
04595     ssize_t write_len = 0;
04596     for (const auto &x : headers) {
04597         std::string s;
04598         s = x.first;
04599         s += ": ";
04600         s += x.second;
04601         s += "\r\n";
04602
04603         auto len = strm.write(s.data(), s.size());
04604         if (len < 0) { return len; }
04605         write_len += len;
04606     }
04607     auto len = strm.write("\r\n");
04608     if (len < 0) { return len; }
04609     write_len += len;
04610     return write_len;
04611 }
```

Граф вызовов:



Граф вызова функции:



5.3.2.103 `write_multipart_ranges_data()`

```

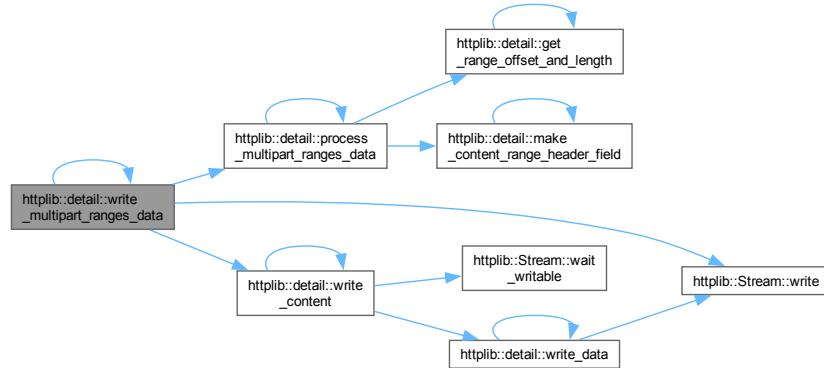
template<typename T>
bool httpplib::detail::write_multipart_ranges_data(
    Stream & strm,
    const Request & req,
    Response & res,
    const std::string & boundary,
    const std::string & content_type,
    size_t content_length,
    const T & is_shutting_down) [inline]
  
```

См. определение в файле `httpplib.h` строка 5454

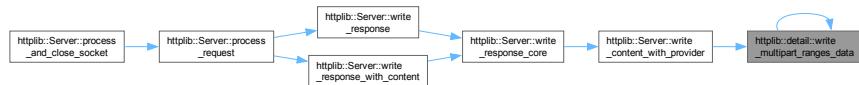
```

05457     return process_multipart_ranges_data(
05458         req, boundary, content_type, content_length,
05459         [&](const std::string &token) { strm.write(token); },
05460         [&](const std::string &token) { strm.write(token); },
05461         [&](size_t offset, size_t length) {
05462             return write_content(strm, res.content_provider_, offset, length,
05463                                 is_shutting_down);
05464         });
05465     });
05466 }
  
```

Граф вызовов:



Граф вызова функции:



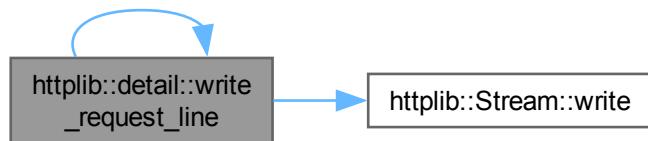
5.3.2.104 `write_request_line()`

```
ssize_t httpplib::detail::write_request_line(
    Stream & strm,
    const std::string & method,
    const std::string & path) [inline]
```

См. определение в файле [httpplib.h](#) строка 4576

```
04577
04578     std::string s = method;
04579     s += " ";
04580     s += path;
04581     s += " HTTP/1.1\r\n";
04582     return strm.write(s.data(), s.size());
04583 }
```

Граф вызовов:



Граф вызова функции:



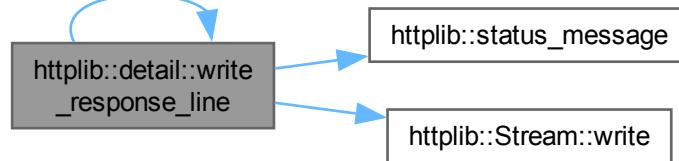
5.3.2.105 `write_response_line()`

```
ssize_t httpplib::detail::write_response_line (
    Stream & strm,
    int status) [inline]
```

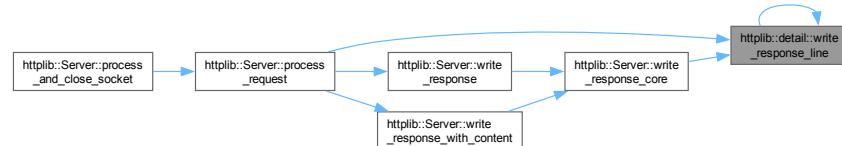
См. определение в файле `httpplib.h` строка [4585](#)

```
04585
04586     std::string s = "HTTP/1.1 ";
04587     s += std::to_string(status);
04588     s += " ";
04589     s += httpplib::status_message(status);
04590     s += "\r\n";
04591     return strm.write(s.data(), s.size());
04592 }
```

Граф вызовов:



Граф вызова функции:



5.4 Пространство имен `httpplib::detail::case_ignore`

Классы

- struct `equal_to`
- struct `hash`

Функции

- `unsigned char to_lower (int c)`
- `bool equal (const std::string &a, const std::string &b)`

5.4.1 Функции

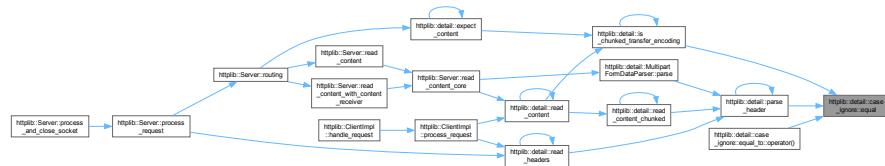
5.4.1.1 `equal()`

```
bool httpplib::detail::case_ignore::equal (
    const std::string & a,
    const std::string & b) [inline]
```

См. определение в файле `httpplib.h` строка [376](#)

```
00376                                     {
00377     return a.size() == b.size() &&
00378         std::equal(a.begin(), a.end(), b.begin(), [](char ca, char cb) {
00379             return to_lower(ca) == to_lower(cb);
00380         });
00381 }
```

Граф вызова функции:



5.4.1.2 `to_lower()`

```
unsigned char httpplib::detail::case_ignore::to_lower (
    int c) [inline]
```

См. определение в файле `httpplib.h` строка [352](#)

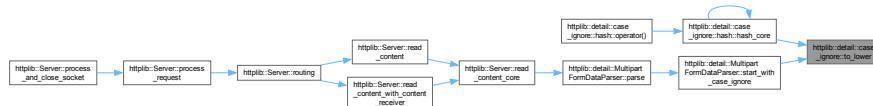
```
00352                                     {
00353     const static unsigned char table[256] = {
00354         0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
00355         15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
00356         30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
00357         45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
00358         60, 61, 62, 63, 64, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
00359         107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121,
00360         122, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
00361         105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
00362         120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
00363         135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
00364         150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
```

```

00365    165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
00366    180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 224, 225, 226,
00367    227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
00368    242, 243, 244, 245, 246, 215, 248, 249, 250, 251, 252, 253, 254, 223, 224,
00369    225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
00370    240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
00371    255,
00372  };
00373  return table[(unsigned char)(char)c];
00374 }

```

Граф вызова функции:



5.5 Пространство имен `httpplib::detail::fields`

Функции

- `bool is_token_char (char c)`
- `bool is_token (const std::string &s)`
- `bool is_field_name (const std::string &s)`
- `bool is_vchar (char c)`
- `bool is_obs_text (char c)`
- `bool is_field_vchar (char c)`
- `bool is_field_content (const std::string &s)`
- `bool is_field_value (const std::string &s)`

5.5.1 Функции

5.5.1.1 `is_field_content()`

```
bool httpplib::detail::fields::is_field_content (
    const std::string & s) [inline]
```

См. определение в файле `httpplib.h` строка 2666

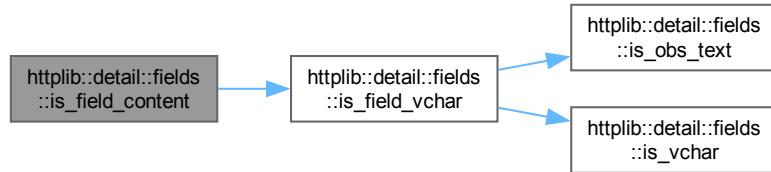
```

02666
02667  if (s.empty()) { return true; }
02668
02669  if (s.size() == 1) {
02670      return is_field_vchar(s[0]);
02671  } else if (s.size() == 2) {
02672      return is_field_vchar(s[0]) && is_field_vchar(s[1]);
02673  } else {
02674      size_t i = 0;
02675
02676      if (!is_field_vchar(s[i])) { return false; }
02677      i++;
02678
02679      while (i < s.size() - 1) {
02680          auto c = s[i++];
02681          if (c == ',' || c == '\t' || is_field_vchar(c)) {
02682              else {
02683                  return false;
02684              }
02685          }
02686
02687      return is_field_vchar(s[i]);

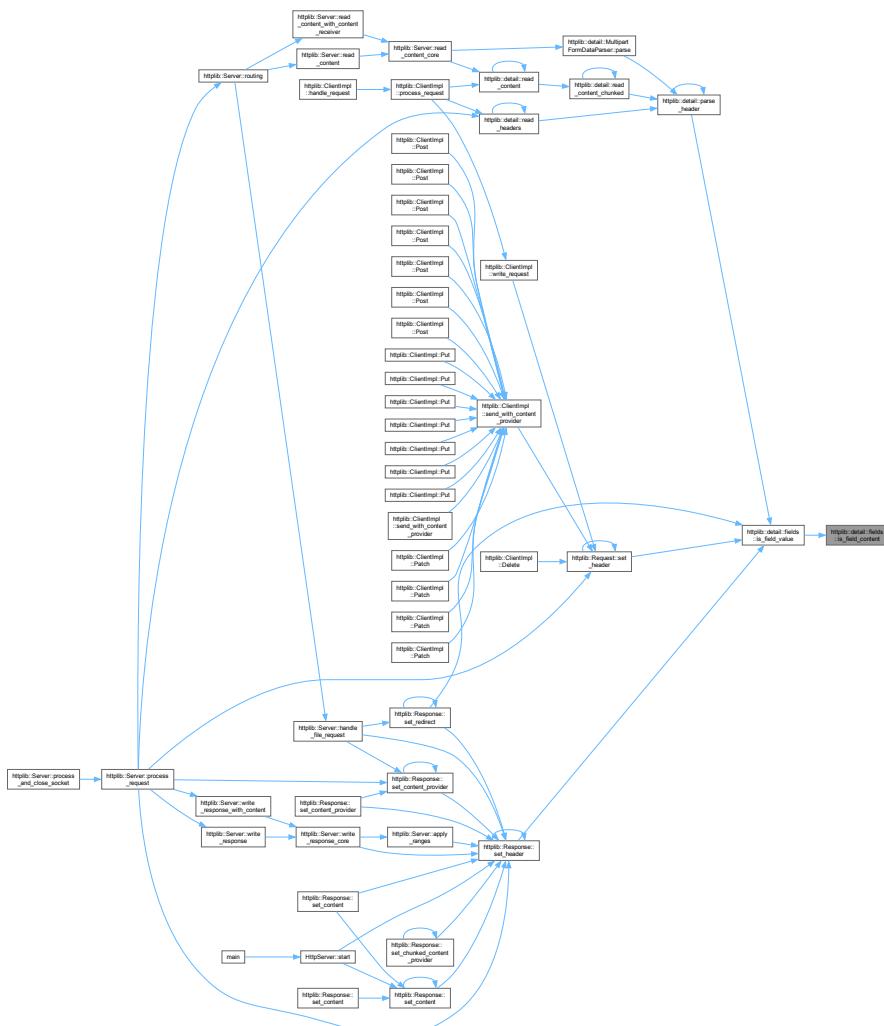
```

```
02688 }
02689 }
```

Граф вызовов:



Граф вызова функции:



5.5.1.2 is_field_name()

```
bool httplib::detail::fields::is_field_name (
```

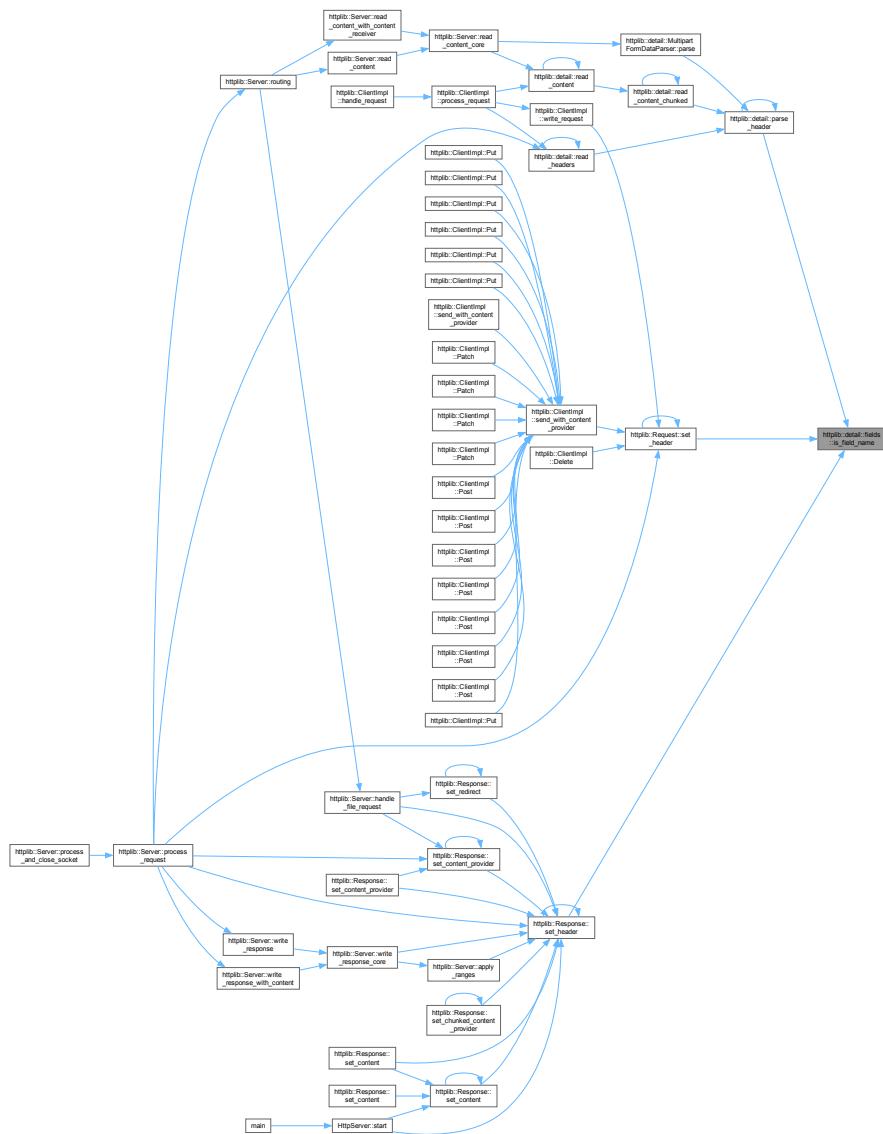
См. определение в файле [httplib.h](#) строка 2658

02658 { return is_token(s); }

Граф вызовов:



Граф вызова функции:

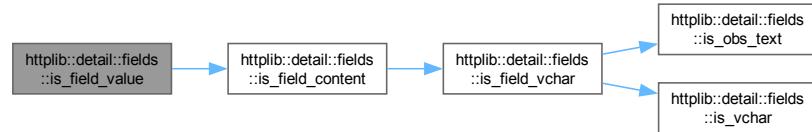


5.5.1.3 is_field_value()

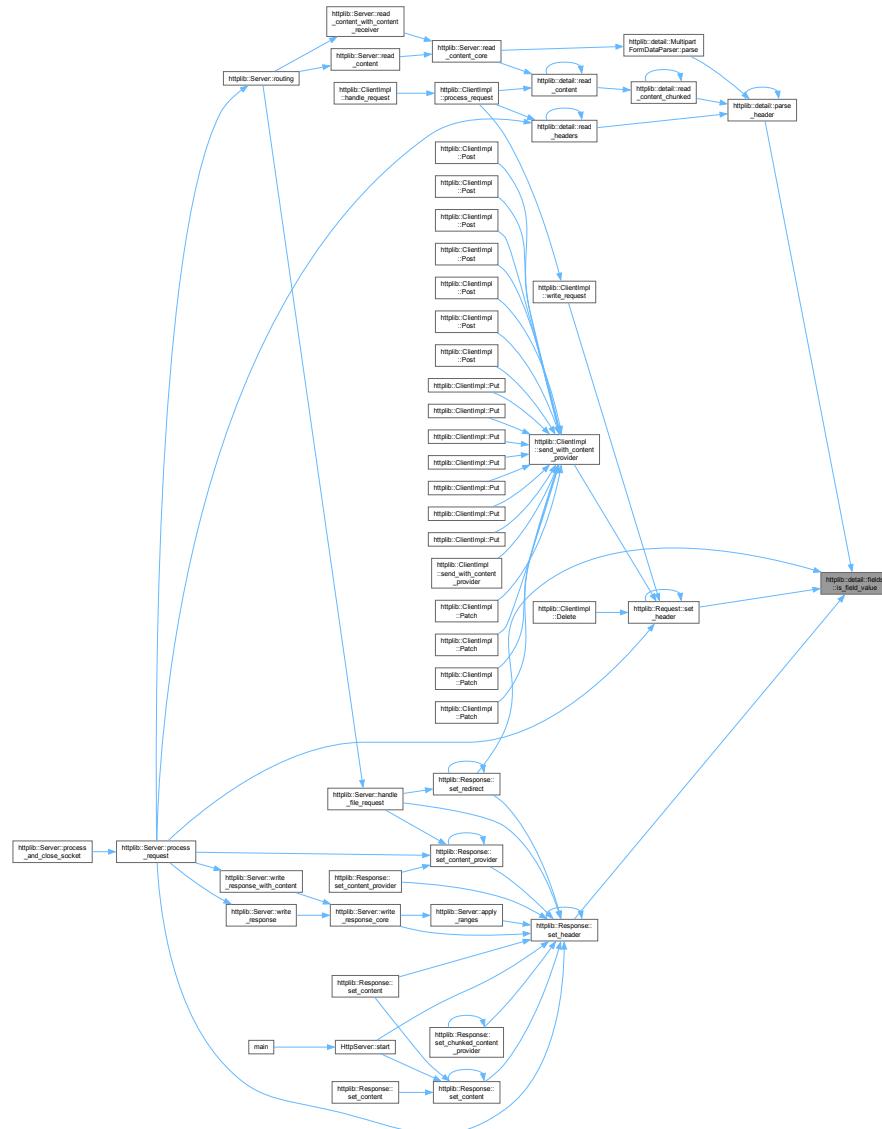
```
bool httplib::detail::fields::is_field_value (
```

См. определение в файле [httpplib.h](#) строка 2691
02691 { return [is_field_content](#)(s); }

Граф вызовов:



Граф вызова функции:



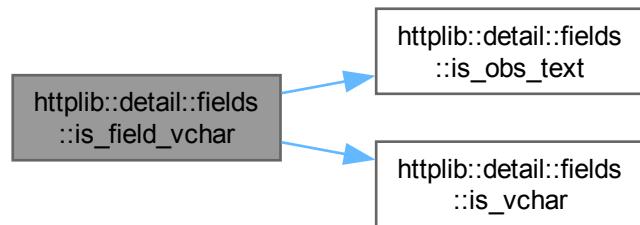
5.5.1.4 is_field_vchar()

```
bool httplib::detail::fields::is_field_vchar (
```

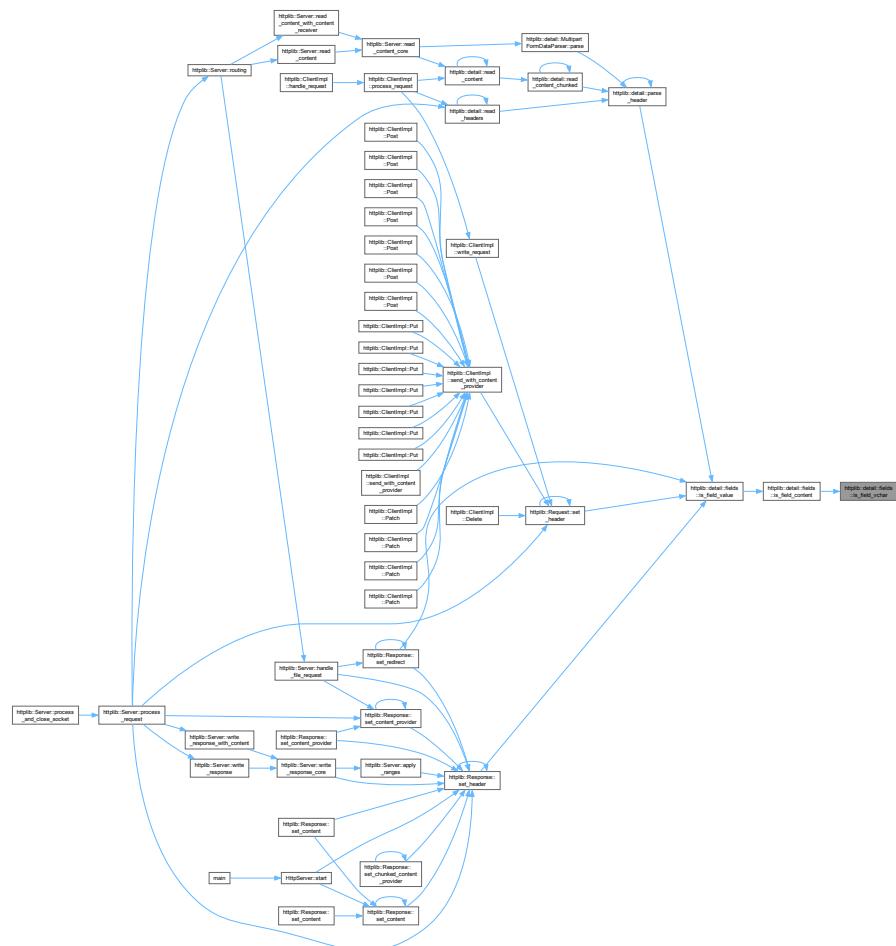
См. определение в файле `httpplib.h` строка 2664

```
02664 { return is_vchar(c) || is_obs_text(c); }
```

Граф вызовов:



Граф вызова функции:



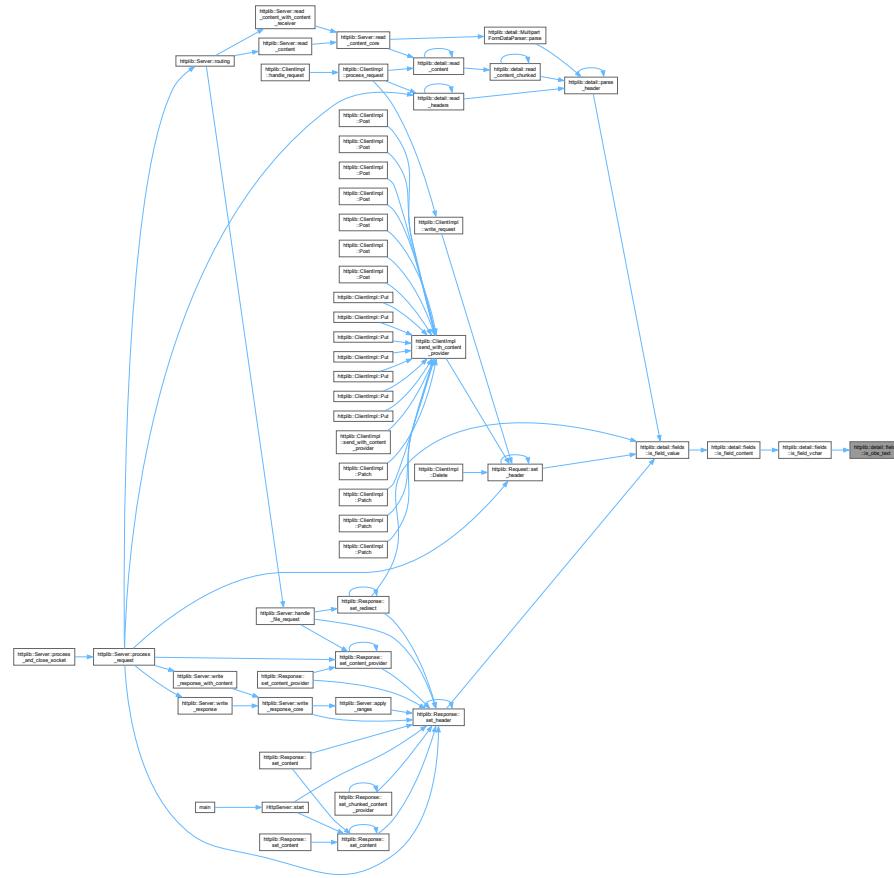
5.5.1.5 `is_obs_text()`

```
bool httpplib::detail::fields::is_obs_text (
    char c) [inline]
```

См. определение в файле `httpplib.h` строка 2662

```
02662 { return 128 <= static_cast<unsigned char>(c); }
```

Граф вызова функции:



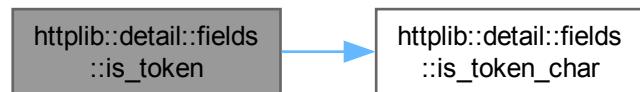
5.5.1.6 `is_token()`

```
bool httpplib::detail::fields::is_token (
    const std::string & s) [inline]
```

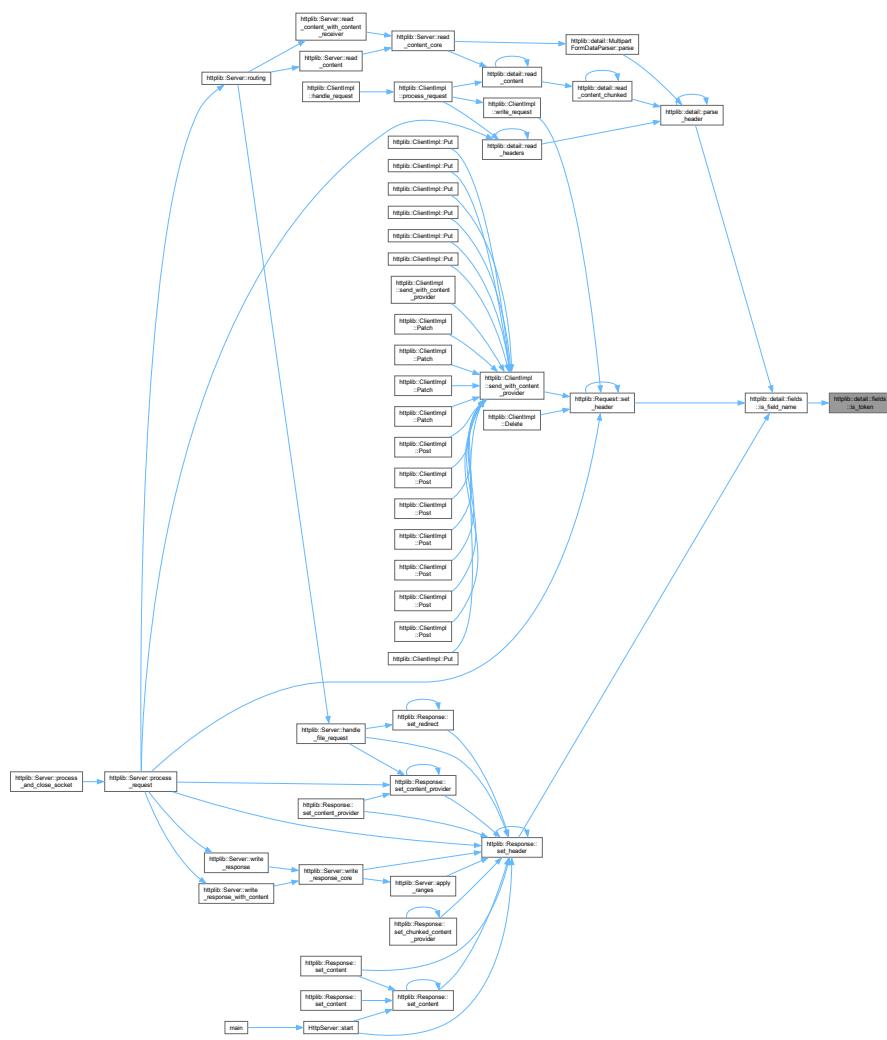
См. определение в файле `httpplib.h` строка 2650

```
02650 if (s.empty()) { return false; }
02651 if (s[0] == ' ') { return true; }
02652 for (auto c : s) {
02653     if (!is_token_char(c)) { return false; }
02654 }
02655 return true;
02656 }
```

Граф вызовов:



Граф вызова функции:



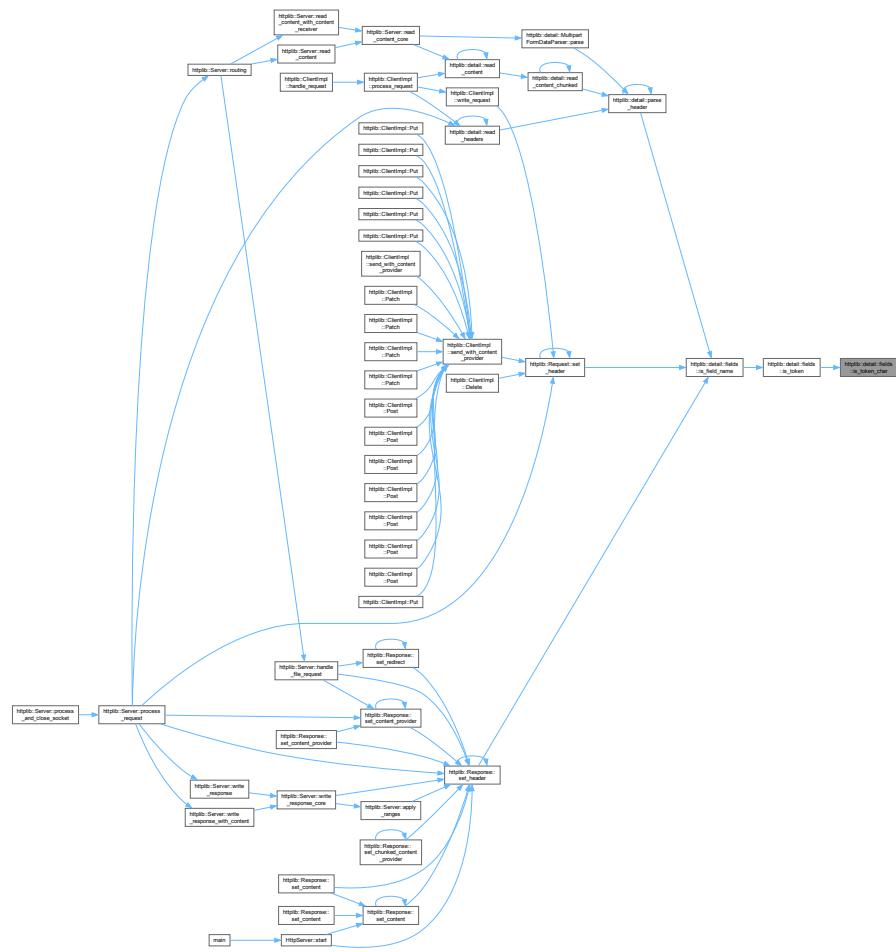
5.5.1.7 is_token char()

```
bool httplib::detail::fields::is_token_char (
```

См. определение в файле [httplib.h](#) строка 2644

```
02644     {
02645     return std::isalnum(c) || c == '!' || c == '#' || c == '$' || c == '%' || 
02646         c == '&' || c == '\n' || c == '*' || c == '+' || c == '_' || 
02647         c == '.' || c == '^' || c == '_' || c == '**' || c == '|' || c == '~';
02648 }
```

Граф вызова функции:



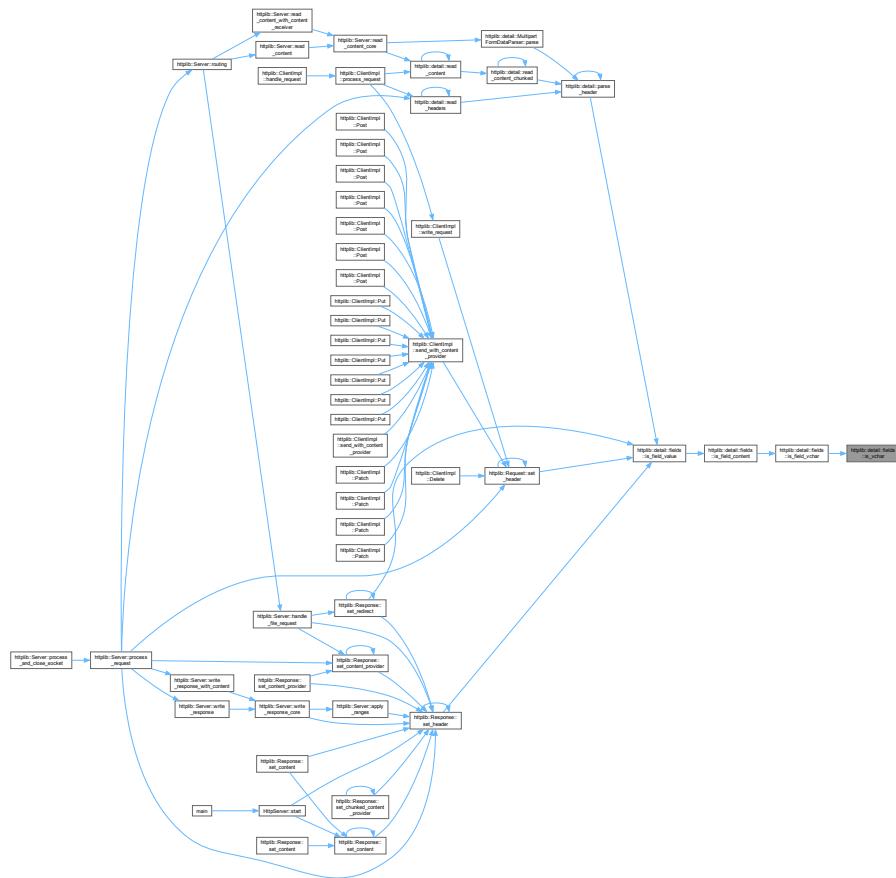
5.5.1.8 is_vchar()

```
bool httplib::detail::fields::is_vchar (
```

См. определение в файле [httpplib.h](#) строка 2660

02660 { **return** c >= 33 && c <= 126; }

Граф вызова функции:



5.6 Пространство имен `httpplib::detail::udl`

Функции

- `constexpr unsigned int operator""_t (const char *s, size_t l)`

5.6.1 Функции

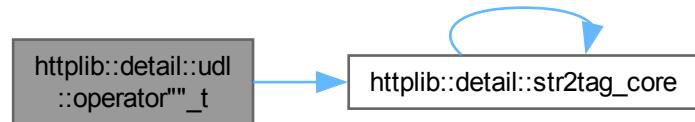
5.6.1.1 operator_t()

```
unsigned int httpplib::detail::udl::operator""_t (
```

См. определение в файле `httpplib.h` строка 3882

```
03882
03883     return str2tag_core(s, l, 0);
03884 }
```

Граф вызовов:



Глава 6

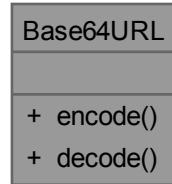
Классы

6.1 Класс Base64URL

Класс для кодирования и декодирования строк в формате [Base64URL](#).

```
#include <Base64URL.h>
```

Граф связей класса Base64URL:



Открытые статические члены

- static std::string [encode](#) (const std::string &input)
Кодирует входную строку в формат [Base64URL](#).
- static std::string [decode](#) (const std::string &input)
Декодирует строку из формата [Base64URL](#) в исходное значение.

6.1.1 Подробное описание

Класс для кодирования и декодирования строк в формате [Base64URL](#).

[Base64URL](#) — это модифицированная версия стандартного Base64, предназначенная для использования в URL и именах файлов.

Отличия от обычного Base64:

- Символы + заменяются на -
- Символы / заменяются на _
- Символы = (для выравнивания) удаляются

Применяется в [JWT](#) для кодирования заголовка и полезной нагрузки.

См. определение в файле [Base64URL.h](#) строка 17

6.1.2 Методы

6.1.2.1 decode()

```
std::string Base64URL::decode (  
    const std::string & input) [static]
```

Декодирует строку из формата [Base64URL](#) в исходное значение.

Алгоритм:

1. Преобразование из [Base64URL](#) обратно в Base64:
 - - → +, _ → /, добавляются символы =, если нужно
2. Расшифровка:
 - Каждые 4 символа → 3 байта оригинальных данных

Аргументы

input	Строка в формате Base64URL
-------	--------------------------------------------

Возвращает

Декодированная строка (обычно — JSON)

См. определение в файле [Base64URL.cpp](#) строка 44

```

00044     {
00045         std::cout << "[Base64URL::decode] Входная строка (Base64URL): " << input << std::endl;
00046
00047         std::string b64 = input;
00048         for (char& c : b64) {
00049             if (c == '+') c = '-';
00050             else if (c == '/') c = '_';
00051         }
00052
00053         while (b64.size() % 4 != 0)
00054             b64 += '=';
00055
00056         std::vector<int> T(256, -1);
00057         for (int i = 0; i < 64; i++) T[base64_chars[i]] = i;
00058
00059         std::string out;
00060         int val = 0, valb = -8;
00061         for (unsigned char c : b64) {
00062             if (T[c] == -1) break;
00063             val = (val << 6) + T[c];
00064             valb += 6;
00065             if (valb >= 0) {
00066                 char ch = char((val >> valb) & 0xFF);
00067                 out.push_back(ch);
00068                 valb -= 8;
00069             }
00070         }
00071
00072         std::cout << "[Base64URL::decode] Декодированная строка: " << out << std::endl;
00073         return out;
00074 }
```

Граф вызова функции:



6.1.2.2 encode()

```
std::string Base64URL::encode (
    const std::string & input) [static]
```

Кодирует входную строку в формат [Base64URL](#).

Алгоритм:

1. Стока сначала кодируется в обычный Base64:
 - Каждый байт добавляется в буфер.
 - Каждые 6 бит буфера превращаются в символ из алфавита Base64.
2. Преобразование в [Base64URL](#):
 - + заменяется на -
 - / заменяется на _
 - = (padding) удаляется

Аргументы

input	Входная строка (обычно — JSON)
-------	--------------------------------

Возвращает

Закодированная строка в формате [Base64URL](#)

См. определение в файле [Base64URL.cpp](#) строка 11

```

00011      {
00012      std::cout << "[Base64URL::encode] Входная строка: " << input << std::endl;
00013
00014      std::string encoded;
00015      int val = 0, valb = -6;
00016      for (unsigned char c : input) {
00017          val = (val << 8) + c;
00018          valb += 8;
00019          while (valb >= 0) {
00020              char ch = base64_chars[(val >> valb) & 0x3F];
00021              encoded.push_back(ch);
00022              valb -= 6;
00023          }
00024      }
00025
00026      if (valb > -6) {
00027          char ch = base64_chars[((val << 8) >> (valb + 8)) & 0x3F];
00028          encoded.push_back(ch);
00029      }
00030
00031 // Преобразование в Base64URL
00032 for (char& c : encoded) {
00033     if (c == '+') c = '_';
00034     else if (c == '/') c = '_';
00035 }
00036
00037 while (!encoded.empty() && encoded.back() == '=')
00038     encoded.pop_back();
00039
00040 std::cout << "[Base64URL::encode] Кодированная строка (Base64URL): " << encoded << std::endl;
00041 return encoded;
00042 }
```

Граф вызова функции:



Объявления и описания членов классов находятся в файлах:

- [Base64URL.h](#)
- [Base64URL.cpp](#)

6.2 Класс BigInt

Класс для работы с большими целыми числами произвольной длины (Big Integer).

```
#include <BigInt.h>
```

Граф связей класса BigInt:

BigInt	
-	digits
-	negative
+	BigInt()
+	toString()
+	toString()
+	operator+()
+	operator-()
+	operator*()
+	operator/()
	и 10 больше...
+	modPow()
+	gcd()
-	trim()
-	compareAbs()

Открытые члены

- [BigInt \(\)](#)

Конструктор по умолчанию. Создаёт число 0.

- [BigInt \(int value\)](#)

Конструктор из целого числа int.

- [BigInt \(const std::string &str\)](#)

Конструктор из строки (десятичное представление).

- [BigInt \(const std::string &str, int base\)](#)

Конструктор из строки с указанной системой счисления.

- [std::string toString \(\) const](#)

Преобразует число в строку в десятичной системе.

- [std::string toString \(int base\) const](#)

Преобразует число в строку в указанной системе счисления.

- **BigInt operator+** (const `BigInt` &other) const
Сложение двух чисел.
- **BigInt operator-** (const `BigInt` &other) const
Вычитание одного числа из другого.
- **BigInt operator*** (const `BigInt` &other) const
Умножение двух чисел.
- **BigInt operator/** (const `BigInt` &other) const
Деление одного числа на другое.
- **BigInt operator%** (const `BigInt` &other) const
Остаток от деления.
- **bool operator<** (const `BigInt` &other) const
Меньше
- **bool operator>** (const `BigInt` &other) const
Больше
- **bool operator==** (const `BigInt` &other) const
Равно
- **bool operator!=** (const `BigInt` &other) const
Не равно
- **bool operator<=** (const `BigInt` &other) const
Меньше или равно
- **bool operator>=** (const `BigInt` &other) const
Больше или равно
- **BigInt operator-** () const
Унарный минус.
- **bool isZero** () const
Проверка на равенство нулю.
- **bool isNegative** () const
Проверка, является ли число отрицательным.

Открытые статические члены

- static `BigInt modPow` (`BigInt` base, `BigInt` exp, const `BigInt` &mod)
Быстрое возведение в степень по модулю.
- static `BigInt gcd` (`BigInt` a, `BigInt` b)
Наибольший общий делитель (НОД).

Закрытые члены

- `void trim` ()
Удаляет ведущие нули.

Закрытые статические члены

- static int `compareAbs` (const `BigInt` &a, const `BigInt` &b)
Сравнение абсолютных значений двух чисел.

Закрытые данные

- std::vector< int > **digits**
Вектор цифр в десятичной системе (младшие разряды первыми)
 - bool **negative** = false
Признак отрицательности числа

6.2.1 Подробное описание

Класс для работы с большими целыми числами произвольной длины (Big Integer).

Реализация поддерживает знаковые числа (отрицательные и положительные), все базовые арифметические операции, операции сравнения, возведение в степень по модулю и вычисление НОД. Используется представление, в котором младшие цифры находятся в начале вектора.

См. определение в файле [BigInt.h](#) строка 12

6.2.2 Конструктор(ы)

6.2.2.1 BigInt() [1/4]

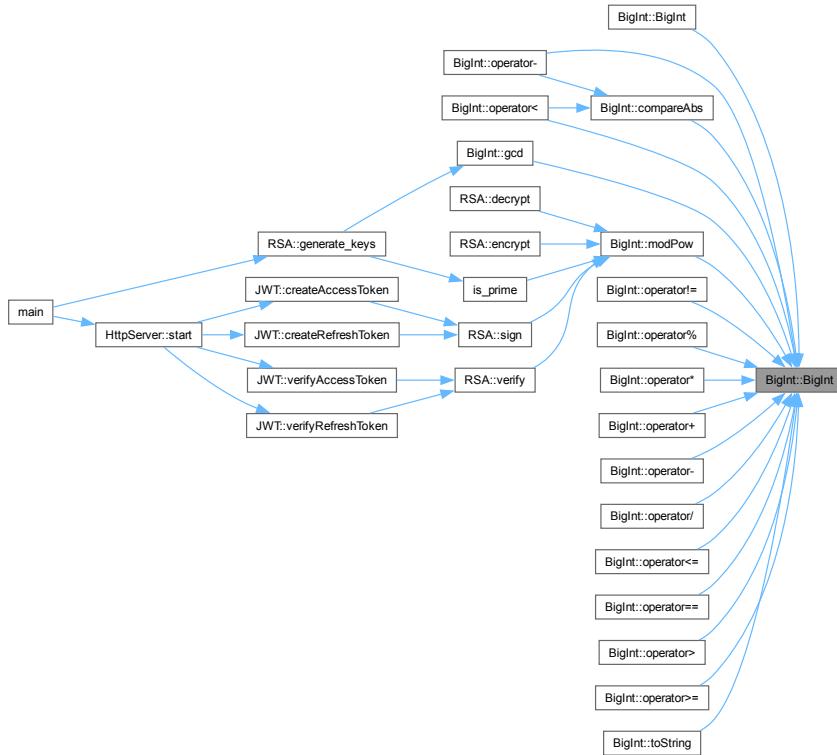
BigInt::BigInt ()

Конструктор по умолчанию. Создаёт число 0.

См. определение в файле [BigInt.cpp](#) строка 6

```
00006 : digits{0}, negative(false) {}
```

Граф вызова функции:



6.2.2.2 BigInt() [2/4]

```
BigInt::BigInt (
    int value)
```

Конструктор из целого числа int.

Аргументы

value	Целое число (может быть отрицательным).
-------	-----------------------------------------

Разбивает число на десятичные цифры и сохраняет их в векторе digits.

См. определение в файле [BigInt.cpp](#) строка 8

```
00008     {
00009     if (value < 0) {
00010         negative = true;
00011         value = -value;
00012     }
00013     do {
00014         digits.push_back(value % 10);
00015         value /= 10;
00016     } while (value > 0);
00017 }
```

6.2.2.3 BigInt() [3/4]

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

Конструктор из строки (десятичное представление).

Аргументы

str	Строка, представляющая число, например "12345" или "-987".
-----	------------------------------------------------------------

Последовательно считывает символы строки, преобразуя их в цифры, начиная с конца строки.

См. определение в файле [BigInt.cpp](#) строка 19

```
00019     {
00020     negative = !str.empty() && str[0] == '-';
00021     int start = negative ? 1 : 0;
00022     for (int i = static_cast<int>(str.length()) - 1; i >= start; --i) {
00023         if (isdigit(str[i]))
00024             digits.push_back(str[i] - '0');
00025     }
00026     if (digits.empty()) digits.push_back(0);
00027     trim();
00028 }
```

Граф вызовов:



6.2.2.4 BigInt() [4/4]

```
BigInt::BigInt (
    const std::string & str,
    int base)
```

Конструктор из строки с указанной системой счисления.

Аргументы

str	Строка с числом.
base	Основание системы счисления (поддерживаются только 10 и 16).

В случае hex преобразует каждую цифру в десятичное значение, умножает на соответствующую степень 16 и накапливает результат.

См. определение в файле [BigInt.cpp](#) строка 30

```
00030     if (base != 10 && base != 16) {
00031         throw std::invalid_argument("Unsupported base");
00032     }
00033
00034     std::string s = str;
00035     negative = false;
00036
00037     if (!s.empty() && s[0] == '-') {
00038         negative = true;
00039         s = s.substr(1);
00040     }
00041
00042     if (base == 10) {
00043         // обычный десятичный парсинг, переиспользуем текущий конструктор
00044         *this = BigInt(s);
00045         if (negative) *this = -(*this);
00046         return;
00047     }
00048
00049     // парсинг hex
00050     BigInt result;
00051     BigInt basePow(1);
00052
00053     for (auto it = s.rbegin(); it != s.rend(); ++it) {
00054         char c = *it;
00055         int value = 0;
00056
00057         if (c >= '0' && c <= '9') value = c - '0';
00058         else if (c >= 'a' && c <= 'f') value = 10 + (c - 'a');
00059         else if (c >= 'A' && c <= 'F') value = 10 + (c - 'A');
00060         else throw std::invalid_argument("Invalid character in hex string");
00061
00062         result = result + basePow * BigInt(value);
00063         basePow = basePow * 16;
00064     }
00065
00066     if (negative) result = -result;
00067
00068     *this = result;
00069
00070 }
```

Граф вызовов:



6.2.3 Методы

6.2.3.1 compareAbs()

```
int BigInt::compareAbs (
    const BigInt & a,
    const BigInt & b) [static], [private]
```

Сравнение абсолютных значений двух чисел.

Аргументы

a	Первое число.
b	Второе число.

Возвращает

-1, если $|a| < |b|$; 0 — если равны; 1 — если $|a| > |b|$.

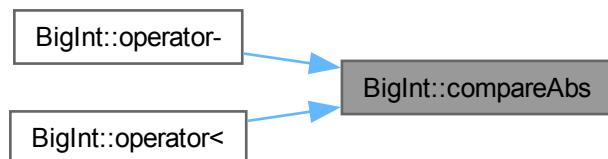
См. определение в файле [BigInt.cpp](#) строка 124

```
00124
00125     if (a.digits.size() != b.digits.size())
00126         return a.digits.size() < b.digits.size() ? -1 : 1;
00127     for (int i = static_cast<int>(a.digits.size()) - 1; i >= 0; --i) {
00128         if (a.digits[i] != b.digits[i])
00129             return a.digits[i] < b.digits[i] ? -1 : 1;
00130     }
00131     return 0;
00132 }
```

Граф вызовов:



Граф вызова функции:



6.2.3.2 gcd()

```
BigInt BigInt::gcd (
    BigInt a,
    BigInt b) [static]
```

Наибольший общий делитель (НОД).

Аргументы

a	Первое число.
b	Второе число.

Возвращает

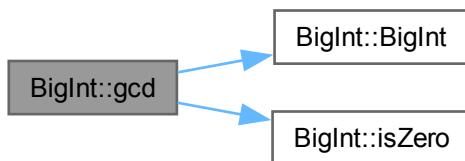
НОД(a, b).

Используется алгоритм Евклида: Пока $b \neq 0$, присваиваем $a = b$, $b = a \% b$.

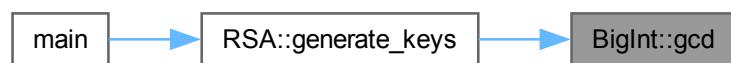
См. определение в файле [BigInt.cpp](#) строка 308

```
00308     {
00309     while (!b.isZero()) {
00310         BigInt temp = b;
00311         b = a % b;
00312         a = temp;
00313     }
00314     return a;
00315 }
```

Граф вызовов:



Граф вызова функции:



6.2.3.3 isNegative()

```
bool BigInt::isNegative () const
```

Проверка, является ли число отрицательным.

См. определение в файле `BigInt.cpp` строка 120

```
00120  
00121    return negative;  
00122 }
```

6.2.3.4 isZero()

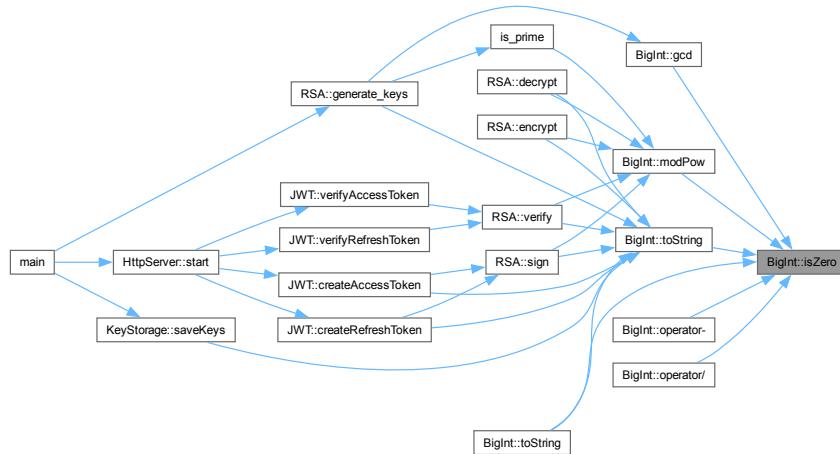
bool BigInt::isZero () const

Проверка на равенство нулю.

См. определение в файле `BigInt.cpp` строка 116

```
00116     {  
00117     return digits.size() == 1 && digits[0] == 0;  
00118 }
```

Граф вызова функции:



6.2.3.5 modPow()

```
BigInt BigInt::modPow (
```

Быстрое возведение в степень по модулю.

Аргументы

base	Основание.
exp	Показатель степени.
mod	Модуль.

Возвращает

$(\text{base}^{\text{exp}}) \% \text{mod}$.

Используется алгоритм "быстрого возведения в степень":

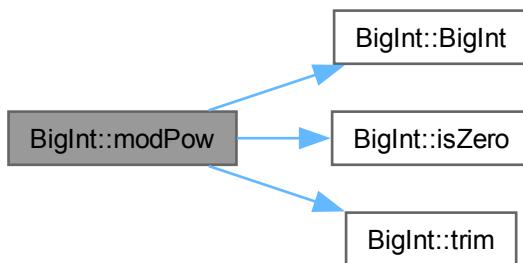
- при каждой итерации проверяется, чётный ли exp ;
- если нечётный — результат $*= \text{base}$;
- затем $\text{base} *= \text{base}$ и $\text{exp} >= 1$ (делится пополам);
- все операции производятся по модулю mod .

См. определение в файле [BigInt.cpp](#) строка 283

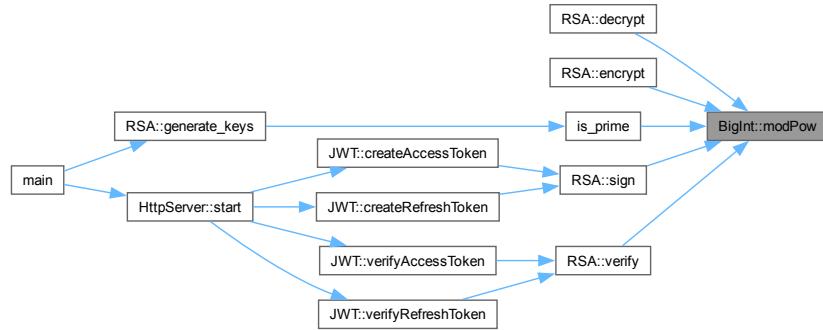
```

00283
00284     base = base % mod;
00285     BigInt result(1);
00286
00287     while (!exp.isZero()) {
00288         if (exp.digits[0] % 2 == 1)
00289             result = (result * base) % mod;
00290         base = (base * base) % mod;
00291
00292         // exp = exp / 2
00293         BigInt half;
00294         half.digits.clear();
00295         int carry = 0;
00296         for (int i = static_cast<int>(exp.digits.size()) - 1; i >= 0; --i) {
00297             int current = carry * 10 + exp.digits[i];
00298             half.digits.insert(half.digits.begin(), current / 2);
00299             carry = current % 2;
00300         }
00301         half.trim();
00302         exp = half;
00303     }
00304
00305     return result;
00306 }
```

Граф вызовов:



Граф вызова функции:



6.2.3.6 operator"!=()

```
bool BigInt::operator!= (
    const BigInt & other) const
```

Не равно

См. определение в файле [BigInt.cpp](#) строка 140

```
00140
00141     return !(*this == other);
00142 }
```

Граф вызовов:



6.2.3.7 operator%()

```
BigInt BigInt::operator% (
    const BigInt & other) const
```

Остаток от деления.

Аргументы

other	Делитель.
-------	-----------

Возвращает

Остаток.

Вычисляется как $*this - (*this / other) * other$.

См. определение в файле [BigInt.cpp](#) строка 278

```
00278     {
00279         return *this - (*this / other) * other;
00280     }
```

Граф вызовов:



6.2.3.8 operator*()

```
BigInt BigInt::operator* (
    const BigInt & other) const
```

Умножение двух чисел.

Аргументы

other	Второй множитель.
-------	-------------------

Возвращает

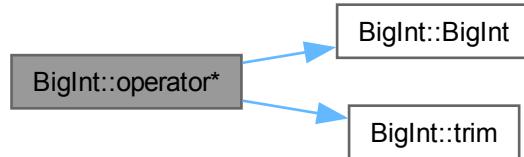
Произведение.

Алгоритм умножения "в столбик": каждый разряд первого числа умножается на каждый разряд второго. Результаты складываются с учётом сдвига и переноса.

См. определение в файле [BigInt.cpp](#) строка 225

```
00225     {
00226         BigInt result;
00227         result.digits.assign(digits.size() + other.digits.size(), 0);
00228         result.negative = negative != other.negative;
00229
00230         for (size_t i = 0; i < digits.size(); ++i) {
00231             int carry = 0;
00232             for (size_t j = 0; j < other.digits.size() || carry; ++j) {
00233                 int64_t cur = result.digits[i + j] +
00234                             digits[i] * 1LL * (j < other.digits.size() ? other.digits[j] : 0) + carry;
00235                 result.digits[i + j] = cur % 10;
00236                 carry = cur / 10;
00237             }
00238         }
00239
00240         result.trim();
00241         return result;
00242     }
```

Граф вызовов:



6.2.3.9 operator+()

```
BigInt BigInt::operator+
    const BigInt & other) const
```

Сложение двух чисел.

Аргументы

other	Второе слагаемое.
-------	-------------------

Возвращает

Сумма чисел.

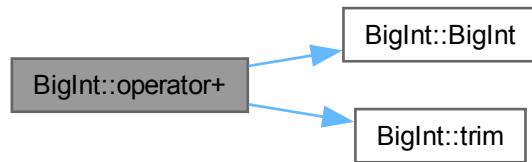
Если знаки равны — складываются поразрядно с учётом переноса. Если знаки разные — используется вычитание с соответствующей сменой знака.

См. определение в файле [BigInt.cpp](#) строка 171

```

00171     {
00172     if (negative == other.negative) {
00173         BigInt result;
00174         result.negative = negative;
00175         result.digits.clear();
00176
00177         int carry = 0;
00178         size_t n = std::max(digits.size(), other.digits.size());
00179         for (size_t i = 0; i < n || carry; ++i) {
00180             int sum = carry;
00181             if (i < digits.size()) sum += digits[i];
00182             if (i < other.digits.size()) sum += other.digits[i];
00183             result.digits.push_back(sum % 10);
00184             carry = sum / 10;
00185         }
00186
00187         result.trim();
00188         return result;
00189     }
00190     return *this - (-other);
00191 }
```

Граф вызовов:



6.2.3.10 operator-() [1/2]

`BigInt` `BigInt::operator- () const`

Унарный минус.

Возвращает

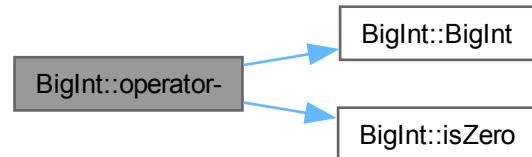
То же самое число, но с противоположным знаком.

См. определение в файле `BigInt.cpp` строка 163

```

00163
00164     BigInt result = *this;
00165     if (!isZero()) result.negative = !negative;
00166     return result;
00167 }
```

Граф вызовов:



6.2.3.11 operator-() [2/2]

`BigInt` `BigInt::operator- (const BigInt & other) const`

Вычитание одного числа из другого.

Аргументы

other	Вычитаемое.
-------	-------------

Возвращает

Разность чисел.

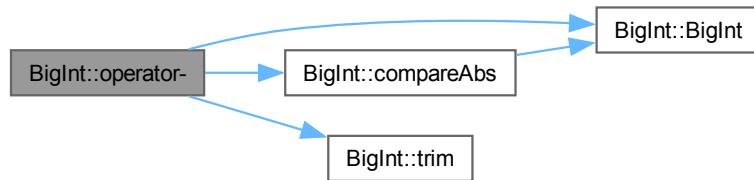
Если знаки разные — происходит сложение. Если одинаковые — сравниваются модули и выполняется поразрядное вычитание с заимствованием.

См. определение в файле [BigInt.cpp](#) строка 193

```

00193     {
00194     if (negative != other.negative) {
00195         return *this + (-other);
00196     }
00197
00198     if (compareAbs(*this, other) < 0) {
00199         BigInt result = other - *this;
00200         result.negative = !negative;
00201         return result;
00202     }
00203
00204     BigInt result;
00205     result.digits.clear();
00206     result.negative = negative;
00207
00208     int borrow = 0;
00209     for (size_t i = 0; i < digits.size(); ++i) {
00210         int diff = digits[i] - borrow;
00211         if (i < other.digits.size()) diff -= other.digits[i];
00212         if (diff < 0) {
00213             diff += 10;
00214             borrow = 1;
00215         } else {
00216             borrow = 0;
00217         }
00218         result.digits.push_back(diff);
00219     }
00220
00221     result.trim();
00222     return result;
00223 }
```

Граф вызовов:



6.2.3.12 operator/()

```

BigInt BigInt::operator/ (
    const BigInt & other) const
  
```

Деление одного числа на другое.

Аргументы

other	Делитель.
-------	-----------

Возвращает

Частное.

Используется алгоритм "деления в столбик": — поразрядно добавляется к текущему значению; — бинарным поиском подбирается максимально возможное значение разряда.

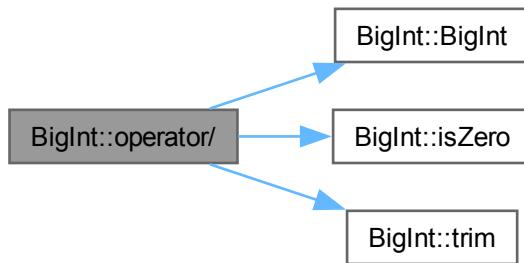
См. определение в файле [BigInt.cpp](#) строка 244

```

00244     {
00245         if (other.isZero()) throw std::domain_error("Division by zero");
00246
00247         BigInt result, current;
00248         result.digits.resize(digits.size());
00249         result.negative = negative != other.negative;
00250
00251         BigInt abs_this = *this; abs_this.negative = false;
00252         BigInt abs_other = other; abs_other.negative = false;
00253
00254         for (int i = static_cast<int>(digits.size()) - 1; i >= 0; --i) {
00255             current.digits.insert(current.digits.begin(), digits[i]);
00256             current.trim();
00257
00258             int x = 0, l = 0, r = 10;
00259             while (l <= r) {
00260                 int m = (l + r) / 2;
00261                 BigInt t = abs_other * BigInt(m);
00262                 if (t <= current) {
00263                     x = m;
00264                     l = m + 1;
00265                 } else {
00266                     r = m - 1;
00267                 }
00268             }
00269
00270             result.digits[i] = x;
00271             current = current - abs_other * BigInt(x);
00272         }
00273
00274         result.trim();
00275         return result;
00276     }

```

Граф вызовов:



6.2.3.13 operator<()

```
bool BigInt::operator< (
    const BigInt & other) const
```

Меньше

См. определение в файле [BigInt.cpp](#) строка 144

```
00144     {
00145         if (negative != other.negative)
00146             return negative;
00147         int cmp = compareAbs(*this, other);
00148         return negative ? cmp > 0 : cmp < 0;
00149     }
```

Граф вызовов:



6.2.3.14 operator<=()

```
bool BigInt::operator<= (
    const BigInt & other) const
```

Меньше или равно

См. определение в файле [BigInt.cpp](#) строка 155

```
00155     {
00156         return !(*this > other);
00157     }
```

Граф вызовов:



6.2.3.15 operator==()

```
bool BigInt::operator== (const BigInt & other) const
```

Равно

См. определение в файле [BigInt.cpp](#) строка 136

```
00136 {  
00137     return negative == other.negative && digits == other.digits;  
00138 }
```

Граф вызовов:



6.2.3.16 operator>()

```
bool BigInt::operator> (const BigInt & other) const
```

Больше

См. определение в файле [BigInt.cpp](#) строка 151

```
00151 {  
00152     return other < *this;  
00153 }
```

Граф вызовов:



6.2.3.17 operator>=()

```
bool BigInt::operator>= (
    const BigInt & other) const
```

Больше или равно

См. определение в файле [BigInt.cpp](#) строка 159

```
00159
00160     return !(*this < other);
00161 }
```

Граф вызовов:



6.2.3.18 toString() [1/2]

```
std::string BigInt::toString () const
```

Преобразует число в строку в десятичной системе.

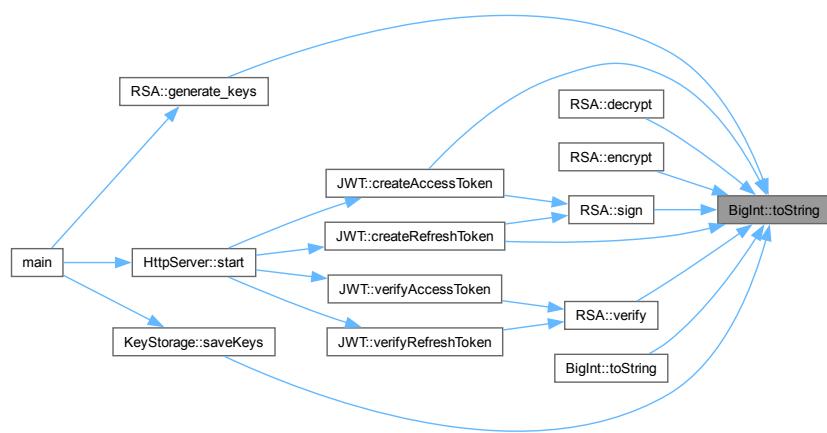
См. определение в файле [BigInt.cpp](#) строка 72

```
00072
00073     std::ostringstream oss;
00074     if (negative && isZero()) oss << "-";
00075     for (auto it = digits.rbegin(); it != digits.rend(); ++it)
00076         oss << *it;
00077     return oss.str();
00078 }
```

Граф вызовов:



Граф вызова функции:



6.2.3.19 `toString()` [2/2]

```
std::string BigInt::toString (
    int base) const
```

Преобразует число в строку в указанной системе счисления.

Аргументы

base	Основание системы счисления (10 или 16).
------	------------------------------------------

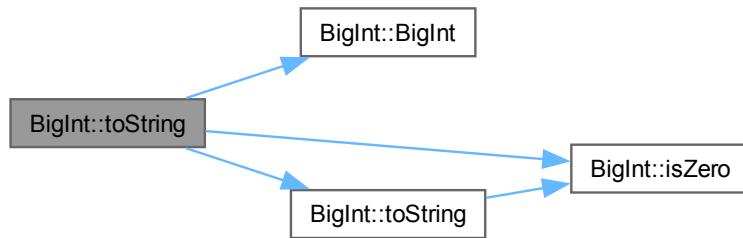
Повторно делит число на основание, записывая остатки.

См. определение в файле [BigInt.cpp](#) строка 80

```

00080
00081     if (base != 10 && base != 16)
00082         throw std::invalid_argument("Unsupported base");
00083
00084     if (isZero()) return "0";
00085
00086     BigInt temp = *this;
00087     temp.negative = false;
00088
00089     std::string result;
00090     BigInt b(base);
00091
00092     while (!temp.isZero()) {
00093         BigInt digit = temp % b;
00094         int d = std::stoi(digit.toString());
00095
00096         char c;
00097         if (d < 10) c = '0' + d;
00098         else c = 'a' + (d - 10);
00099
00100         result += c;
00101         temp = temp / b;
00102     }
00103
00104     if (negative) result += '-';
00105     std::reverse(result.begin(), result.end());
00106     return result;
00107 }
```

Граф вызовов:



6.2.3.20 trim()

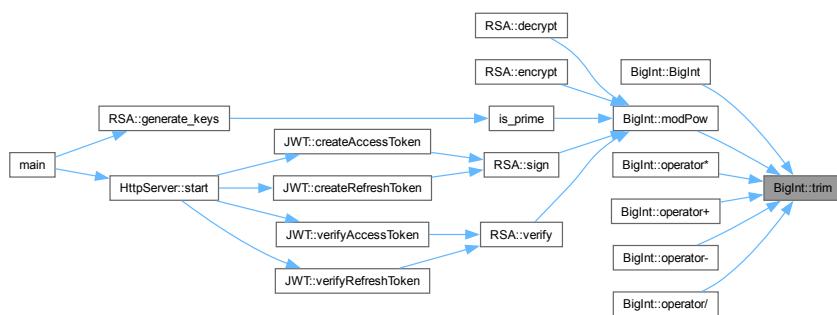
```
void BigInt::trim () [private]
```

Удаляет ведущие нули.

См. определение в файле [BigInt.cpp](#) строка 109

```
00109     {
00110     while (digits.size() > 1 && digits.back() == 0)
00111         digits.pop_back();
00112     if (digits.size() == 1 && digits[0] == 0)
00113         negative = false;
00114 }
```

Граф вызова функции:



6.2.4 Данные класса

6.2.4.1 digits

```
std::vector<int> BigInt::digits [private]
```

Вектор цифр в десятичной системе (младшие разряды первыми)

См. определение в файле [BigInt.h](#) строка 161

6.2.4.2 negative

`bool BigInt::negative = false [private]`

Признак отрицательности числа

См. определение в файле [BigInt.h](#) строка [162](#)

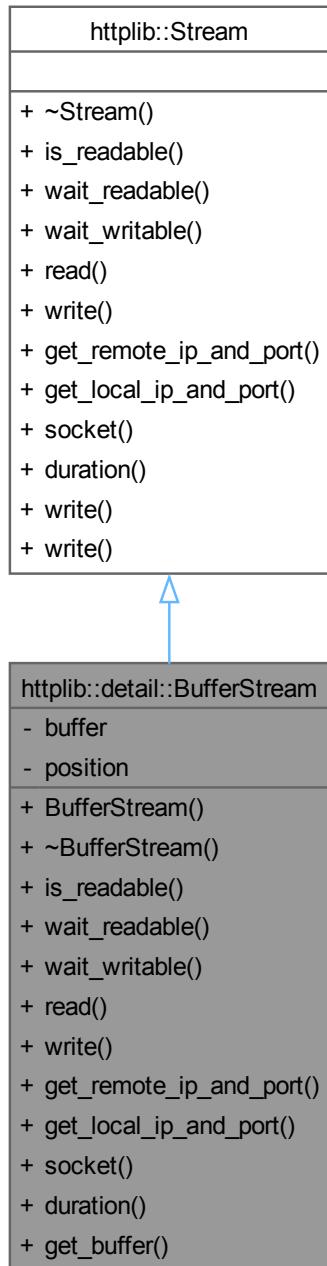
Объявления и описания членов классов находятся в файлах:

- [BigInt.h](#)
- [BigInt.cpp](#)

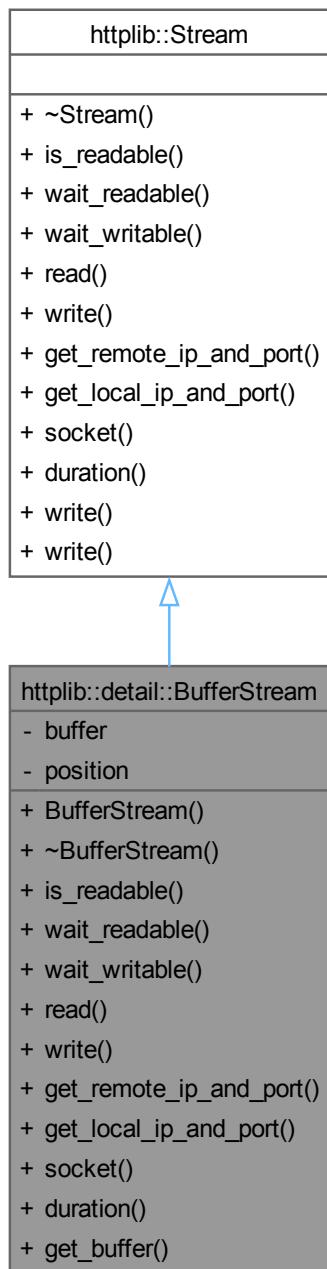
6.3 Класс `httpplib::detail::BufferStream`

```
#include <httpplib.h>
```

Граф наследования:`httpplib::detail::BufferStream`:



Граф связей класса `httpplib::detail::BufferStream`:



Открытые члены

- `BufferStream ()=default`
- `~BufferStream () override=default`
- `bool is_readable () const override`
- `bool wait_readable () const override`
- `bool wait_writable () const override`

- `ssize_t read (char *ptr, size_t size)` override
- `ssize_t write (const char *ptr, size_t size)` override
- `void get_remote_ip_and_port (std::string &ip, int &port) const` override
- `void get_local_ip_and_port (std::string &ip, int &port) const` override
- `socket_t socket () const` override
- `time_t duration () const` override
- `const std::string & get_buffer () const`

Открытые члены унаследованные от [httpplib::Stream](#)

- `virtual ~Stream ()=default`
- `ssize_t write (const char *ptr)`
- `ssize_t write (const std::string &s)`

Закрытые данные

- `std::string buffer`
- `size_t position = 0`

6.3.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 2459

6.3.2 Конструктор(ы)

6.3.2.1 `BufferStream()`

`httpplib::detail::BufferStream::BufferStream () [default]`

6.3.2.2 `~BufferStream()`

`httpplib::detail::BufferStream::~BufferStream () [override], [default]`

6.3.3 Методы

6.3.3.1 `duration()`

`time_t httpplib::detail::BufferStream::duration () const [inline], [override], [virtual]`

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6206

06206 { `return 0;` }

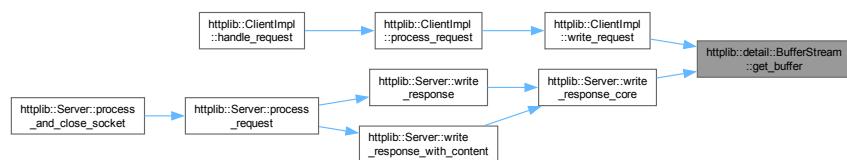
6.3.3.2 `get_buffer()`

```
const std::string & httpplib::detail::BufferStream::get_buffer () const [inline]
```

См. определение в файле [httpplib.h](#) строка [6208](#)

```
06208 { return buffer; }
```

Граф вызова функции:



6.3.3.3 `get_local_ip_and_port()`

```
void httpplib::detail::BufferStream::get_local_ip_and_port (
    std::string & ip,
    int & port) const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка [6201](#)

```
06202 {}
```

6.3.3.4 `get_remote_ip_and_port()`

```
void httpplib::detail::BufferStream::get_remote_ip_and_port (
    std::string & ip,
    int & port) const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка [6198](#)

```
06199 {}
```

6.3.3.5 `is_readable()`

```
bool httpplib::detail::BufferStream::is_readable () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка [6177](#)

```
06177 { return true; }
```

6.3.3.6 `read()`

```
ssize_t httpplib::detail::BufferStream::read (
    char * ptr,
    size_t size) [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6183

```
06183 {
06184 #if defined(_MSC_VER) && _MSC_VER < 1910
06185     auto len_read = buffer._Copy_s(ptr, size, size, position);
06186 #else
06187     auto len_read = buffer.copy(ptr, size, position);
06188 #endif
06189     position += static_cast<size_t>(len_read);
06190     return static_cast<ssize_t>(len_read);
06191 }
```

6.3.3.7 `socket()`

```
socket_t httpplib::detail::BufferStream::socket () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6204

```
06204 { return 0; }
```

6.3.3.8 `wait_readable()`

```
bool httpplib::detail::BufferStream::wait_readable () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6179

```
06179 { return true; }
```

6.3.3.9 `wait_writable()`

```
bool httpplib::detail::BufferStream::wait_writable () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6181

```
06181 { return true; }
```

6.3.3.10 `write()`

```
ssize_t httpplib::detail::BufferStream::write (
    const char * ptr,
    size_t size) [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6193

```
06193 {
06194     buffer.append(ptr, size);
06195     return static_cast<ssize_t>(size);
06196 }
```

6.3.4 Данные класса

6.3.4.1 buffer

```
std::string httpplib::detail::BufferStream::buffer [private]
```

См. определение в файле [httpplib.h](#) строка [2477](#)

6.3.4.2 position

```
size_t httpplib::detail::BufferStream::position = 0 [private]
```

См. определение в файле [httpplib.h](#) строка [2478](#)

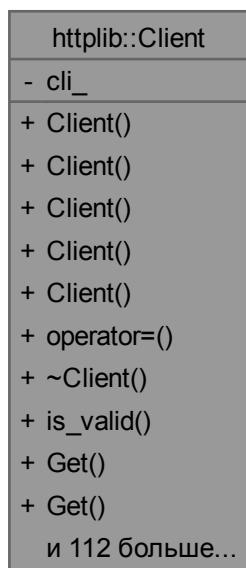
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.4 Класс httpplib::Client

```
#include <httpplib.h>
```

Граф связей класса httpplib::Client:



Открытые члены

- `Client (const std::string &scheme_host_port)`
- `Client (const std::string &scheme_host_port, const std::string &client_cert_path, const std::string &client_key_path)`
- `Client (const std::string &host, int port)`
- `Client (const std::string &host, int port, const std::string &client_cert_path, const std::string &client_key_path)`
- `Client (Client &&) =default`
- `Client & operator= (Client &&) =default`
- `~Client ()`
- `bool is_valid () const`
- `Result Get (const std::string &path)`
- `Result Get (const std::string &path, const Headers &headers)`
- `Result Get (const std::string &path, Progress progress)`
- `Result Get (const std::string &path, const Headers &headers, Progress progress)`
- `Result Get (const std::string &path, ContentReceiver content_receiver)`
- `Result Get (const std::string &path, const Headers &headers, ContentReceiver content_receiver)`
- `Result Get (const std::string &path, ContentReceiver content_receiver, Progress progress)`
- `Result Get (const std::string &path, const Headers &headers, ContentReceiver content_receiver, Progress progress)`
- `Result Get (const std::string &path, ResponseHandler response_handler, ContentReceiver content_receiver)`
- `Result Get (const std::string &path, const Headers &headers, ResponseHandler response_handler, ContentReceiver content_receiver)`
- `Result Get (const std::string &path, const Headers &headers, ResponseHandler response_handler, ContentReceiver content_receiver, Progress progress)`
- `Result Get (const std::string &path, ResponseHandler response_handler, ContentReceiver content_receiver, Progress progress)`
- `Result Get (const std::string &path, const Params ¶ms, const Headers &headers, Progress progress=nullptr)`
- `Result Get (const std::string &path, const Params ¶ms, const Headers &headers, ContentReceiver content_receiver, Progress progress=nullptr)`
- `Result Get (const std::string &path, const Params ¶ms, const Headers &headers, ResponseHandler response_handler, ContentReceiver content_receiver, Progress progress=nullptr)`
- `Result Head (const std::string &path)`
- `Result Head (const std::string &path, const Headers &headers)`
- `Result Post (const std::string &path)`
- `Result Post (const std::string &path, const Headers &headers)`
- `Result Post (const std::string &path, const char *body, size_t content_length, const std::string &content_type)`
- `Result Post (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type)`
- `Result Post (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type, Progress progress)`
- `Result Post (const std::string &path, const std::string &body, const std::string &content_type)`
- `Result Post (const std::string &path, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Post (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type)`
- `Result Post (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Post (const std::string &path, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Post (const std::string &path, ContentProviderWithoutLength content_provider, const std::string &content_type)`

- `Result Post (const std::string &path, const Headers &headers, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Post (const std::string &path, const Headers &headers, ContentProviderWithoutLength content_provider, const std::string &content_type)`
- `Result Post (const std::string &path, const Params ¶ms)`
- `Result Post (const std::string &path, const Headers &headers, const Params ¶ms)`
- `Result Post (const std::string &path, const Headers &headers, const Params ¶ms, Progress progress)`
- `Result Post (const std::string &path, const MultipartFormDataItems &items)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const std::string &boundary)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const MultipartFormDataProviderItems &provider_items)`
- `Result Put (const std::string &path)`
- `Result Put (const std::string &path, const char *body, size_t content_length, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, ContentProviderWithoutLength content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, ContentProviderWithoutLength content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Params ¶ms)`
- `Result Put (const std::string &path, const Headers &headers, const Params ¶ms)`
- `Result Put (const std::string &path, const Headers &headers, const Params ¶ms, Progress progress)`
- `Result Put (const std::string &path, const MultipartFormDataItems &items)`
- `Result Put (const std::string &path, const Headers &headers, const MultipartFormDataItems &items)`
- `Result Put (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const std::string &boundary)`
- `Result Put (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const MultipartFormDataProviderItems &provider_items)`
- `Result Patch (const std::string &path)`
- `Result Patch (const std::string &path, const char *body, size_t content_length, const std::string &content_type)`
- `Result Patch (const std::string &path, const char *body, size_t content_length, const std::string &content_type, Progress progress)`
- `Result Patch (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type)`

- `Result Patch` (const std::string &path, const `Headers` &headers, const char *body, size_t content_length, const std::string &content_type, `Progress` progress)
- `Result Patch` (const std::string &path, const std::string &body, const std::string &content_type)
- `Result Patch` (const std::string &path, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Patch` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type)
- `Result Patch` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Patch` (const std::string &path, size_t content_length, `ContentProvider` content_provider, const std::string &content_type)
- `Result Patch` (const std::string &path, `ContentProviderWithoutLength` content_provider, const std::string &content_type)
- `Result Patch` (const std::string &path, const `Headers` &headers, size_t content_length, `ContentProvider` content_provider, const std::string &content_type)
- `Result Delete` (const std::string &path)
- `Result Delete` (const std::string &path, const `Headers` &headers)
- `Result Delete` (const std::string &path, const char *body, size_t content_length, const std::string &content_type)
- `Result Delete` (const std::string &path, const char *body, size_t content_length, const std::string &content_type, `Progress` progress)
- `Result Delete` (const std::string &path, const `Headers` &headers, const char *body, size_t content_length, const std::string &content_type)
- `Result Delete` (const std::string &path, const `Headers` &headers, const char *body, size_t content_length, const std::string &content_type, `Progress` progress)
- `Result Delete` (const std::string &path, const std::string &body, const std::string &content_type)
- `Result Delete` (const std::string &path, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Options` (const std::string &path)
- `Result Options` (const std::string &path, const `Headers` &headers)
- `bool send` (`Request` &req, `Response` &res, `Error` &error)
- `Result send` (const `Request` &req)
- `void stop()`
- `std::string host()` const
- `int port()` const
- `size_t is_socket_open()` const
- `socket_t socket()` const
- `void set_hostname_addr_map` (std::map< std::string, std::string > addr_map)
- `void set_default_headers` (`Headers` headers)
- `void set_header_writer` (std::function< ssize_t(Stream &, `Headers` &)> const &writer)
- `void set_address_family` (int family)
- `void set_tcp_nodelay` (bool on)
- `void set_socket_options` (`SocketOptions` socket_options)
- `void set_connection_timeout` (time_t sec, time_t usec=0)
- template<class Rep, class Period>
 `void set_connection_timeout` (const std::chrono::duration< Rep, Period > &duration)
- `void set_read_timeout` (time_t sec, time_t usec=0)
- template<class Rep, class Period>
 `void set_read_timeout` (const std::chrono::duration< Rep, Period > &duration)
- `void set_write_timeout` (time_t sec, time_t usec=0)

- template<class Rep, class Period>
void `set_write_timeout` (const std::chrono::duration< Rep, Period > &duration)
- void `set_max_timeout` (time_t msec)
- template<class Rep, class Period>
void `set_max_timeout` (const std::chrono::duration< Rep, Period > &duration)
- void `set_basic_auth` (const std::string &username, const std::string &password)
- void `set_bearer_token_auth` (const std::string &token)
- void `set_keep_alive` (bool on)
- void `set_follow_location` (bool on)
- void `set_url_encode` (bool on)
- void `set_compress` (bool on)
- void `set_decompress` (bool on)
- void `set_interface` (const std::string &intf)
- void `set_proxy` (const std::string &host, int port)
- void `set_proxy_basic_auth` (const std::string &username, const std::string &password)
- void `set_proxy_bearer_token_auth` (const std::string &token)
- void `set_logger` (Logger logger)

Закрытые данные

- std::unique_ptr< `ClientImpl` > `cli_`

6.4.1 Подробное описание

См. определение в файле `httpplib.h` строка 1654

6.4.2 Конструктор(ы)

6.4.2.1 `Client()` [1/5]

```
httpplib::Client::Client (
    const std::string & scheme_host_port) [inline], [explicit]
```

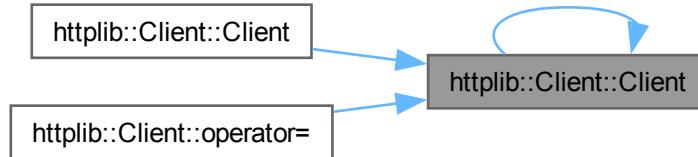
См. определение в файле `httpplib.h` строка 9912

```
09913 : Client(scheme_host_port, std::string(), std::string()) {}
```

Граф вызовов:



Граф вызова функции:



6.4.2.2 Client() [2/5]

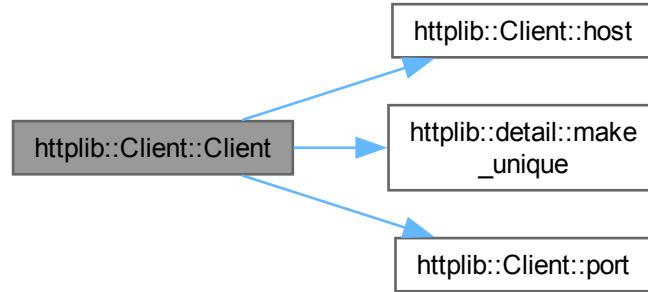
```
httpplib::Client::Client (
    const std::string & scheme_host_port,
    const std::string & client_cert_path,
    const std::string & client_key_path) [inline], [explicit]
```

См. определение в файле `httpplib.h` строка 9915

```

09917
09918     const static std::regex re(
09919         R"((?:([a-z]+):\/\/)?(?:\[([a-fA-F\d:]+)\]|([^\:/?#]+))(?:(\d+))?)");
09920
09921     std::smatch m;
09922     if (std::regex_match(scheme_host_port, m, re)) {
09923         auto scheme = m[1].str();
09924
09925 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
09926         if (!scheme.empty() && (scheme != "http" && scheme != "https")) {
09927             #else
09928                 if (!scheme.empty() && scheme != "http") {
09929             #endif
09930 #ifndef CPPHTTPLIB_NO_EXCEPTIONS
09931                 std::string msg = "The " + scheme + " scheme is not supported.";
09932                 throw std::invalid_argument(msg);
09933             #endif
09934             return;
09935         }
09936
09937         auto is_ssl = scheme == "https";
09938
09939         auto host = m[2].str();
09940         if (host.empty()) { host = m[3].str(); }
09941
09942         auto port_str = m[4].str();
09943         auto port = !port_str.empty() ? std::stoi(port_str) : (is_ssl ? 443 : 80);
09944
09945         if (is_ssl) {
09946 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
09947             cli_ = detail::make_unique<SSLClient>(host, port, client_cert_path,
09948                                                 client_key_path);
09949             is_ssl_ = is_ssl;
09950         #endif
09951     } else {
09952         cli_ = detail::make_unique<ClientImpl>(host, port, client_cert_path,
09953                                                 client_key_path);
09954     }
09955 } else {
09956     // NOTE: Update TEST(UniversalClientImplTest, Ipv6LiteralAddress)
09957     // if port param below changes.
09958     cli_ = detail::make_unique<ClientImpl>(scheme_host_port, 80,
09959                                             client_cert_path, client_key_path);
09960 }
09961 } // namespace detail
  
```

Граф вызовов:

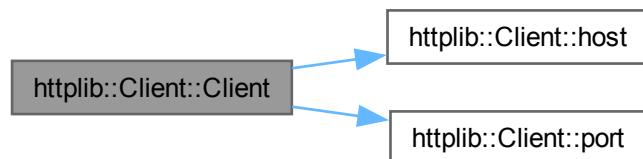


6.4.2.3 Client() [3/5]

```
httpplib::Client::Client (
    const std::string & host,
    int port) [inline], [explicit]
```

См. определение в файле [httpplib.h](#) строка 9963
09964 : `cli_(detail::make_unique<ClientImpl>(host, port)) {}`

Граф вызовов:



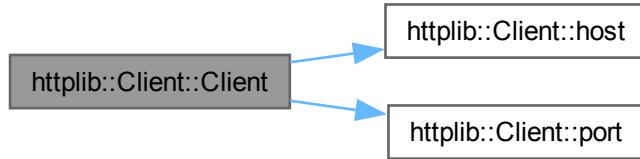
6.4.2.4 Client() [4/5]

```
httpplib::Client::Client (
    const std::string & host,
    int port,
    const std::string & client_cert_path,
    const std::string & client_key_path) [inline], [explicit]
```

См. определение в файле [httpplib.h](#) строка 9966

```
09969 : cli_(detail::make_unique<ClientImpl>(host, port, client_cert_path,
09970           client_key_path)) {}
```

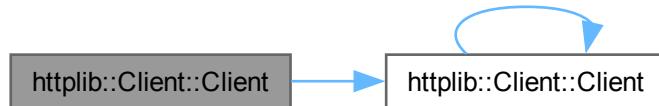
Граф вызовов:



6.4.2.5 Client() [5/5]

```
httplib::Client::Client (
    Client &&) [default]
```

Граф вызовов:



6.4.2.6 ~Client()

```
httplib::Client::~Client () [inline], [default]
```

6.4.3 Методы

6.4.3.1 Delete() [1/10]

```
Result httplib::Client::Delete (
    const std::string & path) [inline]
```

См. определение в файле `httplib.h` строка 10300

```
10300 {
10301     return cli_->Delete(path);
10302 }
```

6.4.3.2 `Delete()` [2/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10306

```
10308 {
10309     return cli_->Delete(path, body, content_length, content_type);
10310 }
```

6.4.3.3 `Delete()` [3/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10311

```
10314 {
10315     return cli_->Delete(path, body, content_length, content_type, progress);
10316 }
```

6.4.3.4 `Delete()` [4/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 10303

```
10303 {
10304     return cli_->Delete(path, headers);
10305 }
```

6.4.3.5 `Delete()` [5/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10317

```
10319 {
10320     return cli_->Delete(path, headers, body, content_length, content_type);
10321 }
```

6.4.3.6 `Delete()` [6/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10322

```
10325     {
10326     return cli_->Delete(path, headers, body, content_length, content_type,
10327                         progress);
10328 }
```

6.4.3.7 `Delete()` [7/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10338

```
10340     {
10341     return cli_->Delete(path, headers, body, content_type);
10342 }
```

6.4.3.8 `Delete()` [8/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10343

```
10346     {
10347     return cli_->Delete(path, headers, body, content_type, progress);
10348 }
```

6.4.3.9 `Delete()` [9/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10329

```
10330     {
10331     return cli_->Delete(path, body, content_type);
10332 }
```

6.4.3.10 `Delete()` [10/10]

```
Result httpplib::Client::Delete (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10333

```
10335 {
10336     return cli_->Delete(path, body, content_type, progress);
10337 }
```

6.4.3.11 `Get()` [1/15]

```
Result httpplib::Client::Get (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 9978

```
09978 { return cli_->Get(path); }
```

6.4.3.12 `Get()` [2/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 9979

```
09979 {
09980     return cli_->Get(path, headers);
09981 }
```

6.4.3.13 `Get()` [3/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httpplib.h` строка 9993

```
09994 {
09995     return cli_->Get(path, headers, std::move(content_receiver));
09996 }
```

6.4.3.14 `Get()` [4/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10001

```
10002 {
10003     return cli_->Get(path, headers, std::move(content_receiver),
10004             std::move(progress));
10005 }
```

6.4.3.15 `Get()` [5/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 9985

```
09986 {
09987     return cli_->Get(path, headers, std::move(progress));
09988 }
```

6.4.3.16 `Get()` [6/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httpplib.h` строка 10012

```
10014 {
10015     return cli_->Get(path, headers, std::move(response_handler),
10016                         std::move(content_receiver));
10017 }
```

6.4.3.17 `Get()` [7/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10024

```
10026 {
10027     return cli_->Get(path, headers, std::move(response_handler),
10028                         std::move(content_receiver), std::move(progress));
10029 }
```

6.4.3.18 `Get()` [8/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    ContentReceiver content_receiver,
    Progress progress = nullptr) [inline]
```

См. определение в файле `httpplib.h` строка 10034

```
10036 {
10037     return cli_->Get(path, params, headers, std::move(content_receiver),
10038                         std::move(progress));
10039 }
```

6.4.3.19 `Get()` [9/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    Progress progress = nullptr) [inline]
```

См. определение в файле `httpplib.h` строка 10030

```
10031 {
10032     return cli_->Get(path, params, headers, std::move(progress));
10033 }
```

6.4.3.20 `Get()` [10/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress = nullptr) [inline]
```

См. определение в файле `httpplib.h` строка 10040

```
10043 {
10044     return cli_->Get(path, params, headers, std::move(response_handler),
10045             std::move(content_receiver), std::move(progress));
10046 }
```

6.4.3.21 `Get()` [11/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httpplib.h` строка 9989

```
09990 {
09991     return cli_->Get(path, std::move(content_receiver));
09992 }
```

6.4.3.22 `Get()` [12/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 9997

```
09998 {
09999     return cli_->Get(path, std::move(content_receiver), std::move(progress));
10000 }
```

6.4.3.23 `Get()` [13/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 9982

```
09982 {  
09983     return cli_->Get(path, std::move(progress));  
09984 }
```

6.4.3.24 `Get()` [14/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    ResponseHandler response_handler,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httpplib.h` строка 10006

```
10008 {  
10009     return cli_->Get(path, std::move(response_handler),
10010             std::move(content_receiver));  
10011 }
```

6.4.3.25 `Get()` [15/15]

```
Result httpplib::Client::Get (
    const std::string & path,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10018

```
10020 {  
10021     return cli_->Get(path, std::move(response_handler),
10022             std::move(content_receiver), std::move(progress));  
10023 }
```

6.4.3.26 `Head()` [1/2]

```
Result httpplib::Client::Head (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 10048

```
10048 { return cli_->Head(path); }
```

6.4.3.27 `Head()` [2/2]

```
Result httpplib::Client::Head (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 10049

```
10049 {  
10050     return cli_->Head(path, headers);  
10051 }
```

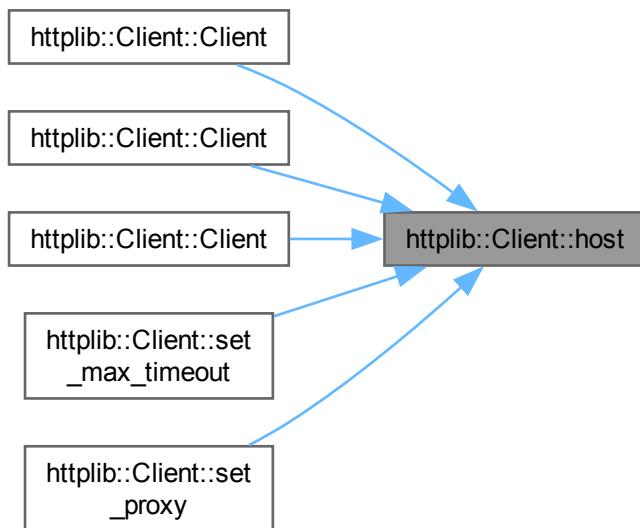
6.4.3.28 `host()`

```
std::string httpplib::Client::host () const [inline]
```

См. определение в файле [httpplib.h](#) строка 10364

```
10364 { return cli_->host(); }
```

Граф вызова функции:

6.4.3.29 `is_socket_open()`

```
size_t httpplib::Client::is_socket_open () const [inline]
```

См. определение в файле [httpplib.h](#) строка 10368

```
10368 { return cli_->is_socket_open(); }
```

6.4.3.30 `is_valid()`

```
bool httpplib::Client::is_valid () const [inline]
```

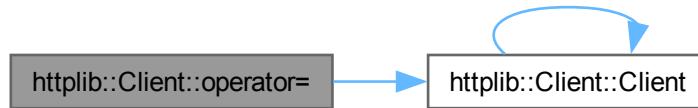
См. определение в файле [httpplib.h](#) строка 9974

```
9974
9975 return cli_- != nullptr && cli_->is_valid();
9976 }
```

6.4.3.31 `operator=()`

```
Client & httpplib::Client::operator= (
    Client &&) [default]
```

Граф вызовов:

6.4.3.32 `Options() [1/2]`

```
Result httpplib::Client::Options (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 10349

```
10349
10350     return cli_->Options(path);
10351 }
```

6.4.3.33 `Options() [2/2]`

```
Result httpplib::Client::Options (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 10352

```
10352
10353     return cli_->Options(path, headers);
10354 }
```

6.4.3.34 `Patch() [1/13]`

```
Result httpplib::Client::Patch (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 10231

```
10231
10232     return cli_->Patch(path);
10233 }
```

6.4.3.35 `Patch()` [2/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10234

```
10236 {
10237     return cli_->Patch(path, body, content_length, content_type);
10238 }
```

6.4.3.36 `Patch()` [3/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10239

```
10242 {
10243     return cli_->Patch(path, body, content_length, content_type, progress);
10244 }
```

6.4.3.37 `Patch()` [4/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10245

```
10247 {
10248     return cli_->Patch(path, headers, body, content_length, content_type);
10249 }
```

6.4.3.38 `Patch()` [5/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10250

```
10253 {
10254     return cli_->Patch(path, headers, body, content_length, content_type,
10255             progress);
10256 }
```

6.4.3.39 `Patch()` [6/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10266

```
10268 {
10269     return cli_->Patch(path, headers, body, content_type);
10270 }
```

6.4.3.40 `Patch()` [7/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10271

```
10274 {
10275     return cli_->Patch(path, headers, body, content_type, progress);
10276 }
```

6.4.3.41 `Patch()` [8/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10295

```
10297 {
10298     return cli_->Patch(path, headers, std::move(content_provider), content_type);
10299 }
```

6.4.3.42 `Patch()` [9/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const Headers & headers,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10288

```
10291 {
10292     return cli_->Patch(path, headers, content_length, std::move(content_provider),
10293                         content_type);
10294 }
```

6.4.3.43 Patch() [10/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10257

```
10258 {
10259     return cli_->Patch(path, body, content_type);
10260 }
```

6.4.3.44 Patch() [11/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10261

```
10262 {
10263     return cli_->Patch(path, body, content_type, progress);
10264 }
10265 }
```

6.4.3.45 Patch() [12/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10283

```
10284 {
10285     return cli_->Patch(path, std::move(content_provider), content_type);
10286 }
10287 }
```

6.4.3.46 Patch() [13/13]

```
Result httpplib::Client::Patch (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10277

```
10278 {
10279     return cli_->Patch(path, content_length, std::move(content_provider),
10280                         content_type);
10281 }
10282 }
```

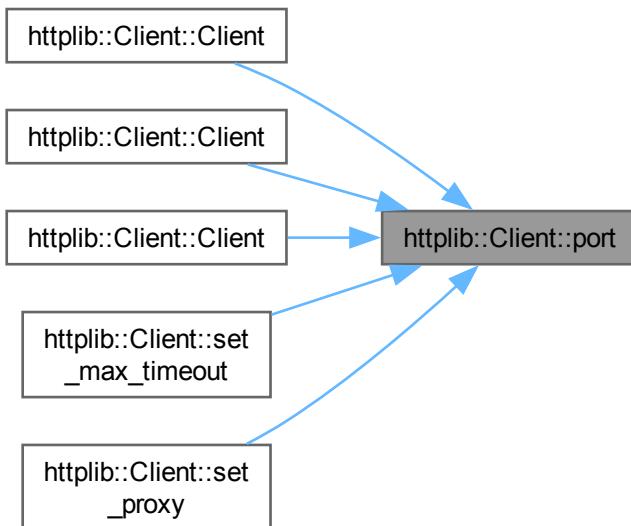
6.4.3.47 `port()`

```
int httpplib::Client::port () const [inline]
```

См. определение в файле `httpplib.h` строка 10366

```
10366 { return cli_->port(); }
```

Граф вызова функции:

6.4.3.48 `Post()` [1/20]

```
Result httpplib::Client::Post (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 10053

```
10053 { return cli_->Post(path); }
```

6.4.3.49 `Post()` [2/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10057

```
10059
10060 { return cli_->Post(path, body, content_length, content_type);
10061 }
```

6.4.3.50 `Post()` [3/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 10054

```
10054                                     {
10055     return cli_->Post(path, headers);
10056 }
```

6.4.3.51 `Post()` [4/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10062

```
10064                                     {
10065     return cli_->Post(path, headers, body, content_length, content_type);
10066 }
```

6.4.3.52 `Post()` [5/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10067

```
10069                                     {
10070     return cli_->Post(path, headers, body, content_length, content_type,
10071                     progress);
10072 }
```

6.4.3.53 `Post()` [6/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле `httpplib.h` строка 10129

```
10130                                     {
10131     return cli_->Post(path, headers, items);
10132 }
```

6.4.3.54 `Post()` [7/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const MultipartFormDataProviderItems & provider_items) [inline]
```

См. определение в файле `httpplib.h` строка 10139

```
10141     {
10142     return cli_->Post(path, headers, items, provider_items);
10143 }
```

6.4.3.55 `Post()` [8/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const std::string & boundary) [inline]
```

См. определение в файле `httpplib.h` строка 10133

```
10135     {
10136     return cli_->Post(path, headers, items, boundary);
10137 }
```

6.4.3.56 `Post()` [9/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const Params & params) [inline]
```

См. определение в файле `httpplib.h` строка 10117

```
10118     {
10119     return cli_->Post(path, headers, params);
10120 }
```

6.4.3.57 `Post()` [10/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const Params & params,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10121

```
10122     {
10123     return cli_->Post(path, headers, params, progress);
10124 }
```

6.4.3.58 `Post()` [11/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10081

```
10083     {
10084     return cli_->Post(path, headers, body, content_type);
10085 }
```

6.4.3.59 `Post()` [12/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10086

```
10088     {
10089     return cli_->Post(path, headers, body, content_type, progress);
10090 }
```

6.4.3.60 `Post()` [13/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10109

```
10111     {
10112     return cli_->Post(path, headers, std::move(content_provider), content_type);
10113 }
```

6.4.3.61 `Post()` [14/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Headers & headers,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10102

```
10105     {
10106     return cli_->Post(path, headers, content_length, std::move(content_provider),
10107                         content_type);
10108 }
```

6.4.3.62 `Post()` [15/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле `httpplib.h` строка 10125

```
10126 {  
10127     return cli_->Post(path, items);  
10128 }
```

6.4.3.63 `Post()` [16/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const Params & params) [inline]
```

См. определение в файле `httpplib.h` строка 10114

```
10114 {  
10115     return cli_->Post(path, params);  
10116 }
```

6.4.3.64 `Post()` [17/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10073

```
10074 {  
10075     return cli_->Post(path, body, content_type);  
10076 }
```

6.4.3.65 `Post()` [18/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10077

```
10078 {  
10079     return cli_->Post(path, body, content_type, progress);  
10080 }
```

6.4.3.66 `Post()` [19/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10097

```
10099 {  
10100     return cli_->Post(path, std::move(content_provider), content_type);  
10101 }
```

6.4.3.67 `Post()` [20/20]

```
Result httpplib::Client::Post (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10091

```
10093     {
10094     return cli_->Post(path, content_length, std::move(content_provider),
10095             content_type);
10096 }
```

6.4.3.68 `Put()` [1/19]

```
Result httpplib::Client::Put (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 10144

```
10144 { return cli_->Put(path); }
```

6.4.3.69 `Put()` [2/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10145

```
10147     {
10148     return cli_->Put(path, body, content_length, content_type);
10149 }
```

6.4.3.70 `Put()` [3/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10150

```
10152     {
10153     return cli_->Put(path, headers, body, content_length, content_type);
10154 }
```

6.4.3.71 `Put()` [4/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле [httpplib.h](#) строка 10155

```
10157     {
10158     return cli_->Put(path, headers, body, content_length, content_type, progress);
10159 }
```

6.4.3.72 `Put()` [5/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле [httpplib.h](#) строка 10216

```
10217     {
10218     return cli_->Put(path, headers, items);
10219 }
```

6.4.3.73 `Put()` [6/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const MultipartFormDataProviderItems & provider_items) [inline]
```

См. определение в файле [httpplib.h](#) строка 10226

```
10228     {
10229     return cli_->Put(path, headers, items, provider_items);
10230 }
```

6.4.3.74 `Put()` [7/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const std::string & boundary) [inline]
```

См. определение в файле [httpplib.h](#) строка 10220

```
10222     {
10223     return cli_->Put(path, headers, items, boundary);
10224 }
```

6.4.3.75 `Put()` [8/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const Params & params) [inline]
```

См. определение в файле `httpplib.h` строка 10204

```
10205     {
10206     return cli_->Put(path, headers, params);
10207 }
```

6.4.3.76 `Put()` [9/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const Params & params,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10208

```
10209     {
10210     return cli_->Put(path, headers, params, progress);
10211 }
```

6.4.3.77 `Put()` [10/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10168

```
10170     {
10171     return cli_->Put(path, headers, body, content_type);
10172 }
```

6.4.3.78 `Put()` [11/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10173

```
10175     {
10176     return cli_->Put(path, headers, body, content_type, progress);
10177 }
```

6.4.3.79 `Put()` [12/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10196

```
10198 {
10199     return cli_->Put(path, headers, std::move(content_provider), content_type);
10200 }
```

6.4.3.80 `Put()` [13/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Headers & headers,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10189

```
10192 {
10193     return cli_->Put(path, headers, content_length, std::move(content_provider),
10194             content_type);
10195 }
```

6.4.3.81 `Put()` [14/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле `httpplib.h` строка 10212

```
10213 {
10214     return cli_->Put(path, items);
10215 }
```

6.4.3.82 `Put()` [15/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const Params & params) [inline]
```

См. определение в файле `httpplib.h` строка 10201

```
10201 {
10202     return cli_->Put(path, params);
10203 }
```

6.4.3.83 `Put()` [16/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10160

```
10161 {
10162     return cli_->Put(path, body, content_type);
10163 }
```

6.4.3.84 `Put()` [17/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 10164

```
10165 {
10166     return cli_->Put(path, body, content_type, progress);
10167 }
```

6.4.3.85 `Put()` [18/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10184

```
10186 {
10187     return cli_->Put(path, std::move(content_provider), content_type);
10188 }
```

6.4.3.86 `Put()` [19/19]

```
Result httpplib::Client::Put (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 10178

```
10180 {
10181     return cli_->Put(path, content_length, std::move(content_provider),
10182                         content_type);
10183 }
```

6.4.3.87 `send()` [1/2]

```
Result httpplib::Client::send (
    const Request & req) [inline]
```

См. определение в файле `httpplib.h` строка 10360

```
10360 { return cli_->send(req); }
```

6.4.3.88 `send()` [2/2]

```
bool httpplib::Client::send (
    Request & req,
    Response & res,
    Error & error) [inline]
```

См. определение в файле `httpplib.h` строка 10356

```
10356 {
10357     return cli_->send(req, res, error);
10358 }
```

6.4.3.89 `set_address_family()`

```
void httpplib::Client::set_address_family (
    int family) [inline]
```

См. определение в файле [httpplib.h](#) строка 10386

```
10386 {  
10387   cli_->set_address_family(family);  
10388 }
```

6.4.3.90 `set_basic_auth()`

```
void httpplib::Client::set_basic_auth (
    const std::string & username,
    const std::string & password) [inline]
```

См. определение в файле [httpplib.h](#) строка 10408

```
10409 {  
10410   cli_->set_basic_auth(username, password);  
10411 }
```

6.4.3.91 `set_bearer_token_auth()`

```
void httpplib::Client::set_bearer_token_auth (
    const std::string & token) [inline]
```

См. определение в файле [httpplib.h](#) строка 10412

```
10412 {  
10413   cli_->set_bearer_token_auth(token);  
10414 }
```

6.4.3.92 `set_compress()`

```
void httpplib::Client::set_compress (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 10429

```
10429 { cli_->set_compress(on); }
```

6.4.3.93 `set_connection_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::Client::set_connection_timeout (
    const std::chrono::duration<Rep, Period> & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2323

```
02324 {  
02325   cli_->set_connection_timeout(duration);  
02326 }
```

6.4.3.94 `set_connection_timeout()` [2/2]

```
void httpplib::Client::set_connection_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле `httpplib.h` строка 10396

```
10396 {  
10397     cli_->set_connection_timeout(sec, usec);  
10398 }
```

6.4.3.95 `set_decompress()`

```
void httpplib::Client::set_decompress (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 10431

```
10431 { cli_->set_decompress(on); }
```

6.4.3.96 `set_default_headers()`

```
void httpplib::Client::set_default_headers (
    Headers headers) [inline]
```

См. определение в файле `httpplib.h` строка 10377

```
10377 {  
10378     cli_->set_default_headers(std::move(headers));  
10379 }
```

6.4.3.97 `set_follow_location()`

```
void httpplib::Client::set_follow_location (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 10423

```
10423 {  
10424     cli_->set_follow_location(on);  
10425 }
```

6.4.3.98 `set_header_writer()`

```
void httpplib::Client::set_header_writer (
    std::function< ssize_t(Stream &, Headers &) > const & writer) [inline]
```

См. определение в файле `httpplib.h` строка 10381

```
10382 {  
10383     cli_->set_header_writer(writer);  
10384 }
```

6.4.3.99 `set_hostname_addr_map()`

```
void httpplib::Client::set_hostname_addr_map (
    std::map< std::string, std::string > addr_map) [inline]
```

См. определение в файле [httpplib.h](#) строка 10373

```
10373     {
10374     cli_->set_hostname_addr_map(std::move(addr_map));
10375 }
```

6.4.3.100 `set_interface()`

```
void httpplib::Client::set_interface (
    const std::string & intf) [inline]
```

См. определение в файле [httpplib.h](#) строка 10433

```
10433     {
10434     cli_->set_interface(intf);
10435 }
```

6.4.3.101 `set_keep_alive()`

```
void httpplib::Client::set_keep_alive (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 10422

```
10422 { cli_->set_keep_alive(on); }
```

6.4.3.102 `set_logger()`

```
void httpplib::Client::set_logger (
    Logger logger) [inline]
```

См. определение в файле [httpplib.h](#) строка 10469

```
10469     {
10470     cli_->set_logger(std::move(logger));
10471 }
```

6.4.3.103 `set_max_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::Client::set_max_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

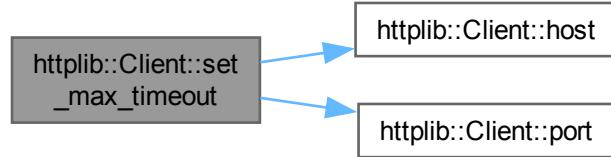
См. определение в файле [httpplib.h](#) строка 2342

```
02342     {
02343     cli_->set_max_timeout(duration);
02344 }
```

6.4.3.104 `set_max_timeout()` [2/2]

```
void httpplib::Client::set_max_timeout (
    time_t msec)
```

Граф вызовов:

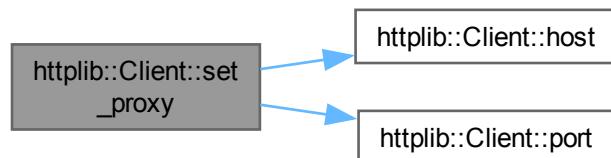
6.4.3.105 `set_proxy()`

```
void httpplib::Client::set_proxy (
    const std::string & host,
    int port) [inline]
```

См. определение в файле [httpplib.h](#) строка 10437

```
10437     {
10438     cli_->set_proxy(host, port);
10439 }
```

Граф вызовов:

6.4.3.106 `set_proxy_basic_auth()`

```
void httpplib::Client::set_proxy_basic_auth (
    const std::string & username,
    const std::string & password) [inline]
```

См. определение в файле [httpplib.h](#) строка 10440

```
10441     {
10442     cli_->set_proxy_basic_auth(username, password);
10443 }
```

6.4.3.107 `set_proxy_bearer_token_auth()`

```
void httpplib::Client::set_proxy_bearer_token_auth (
    const std::string & token) [inline]
```

См. определение в файле [httpplib.h](#) строка [10444](#)

```
10444 {  
10445     cli_->set_proxy_bearer_token_auth(token);  
10446 }
```

6.4.3.108 `set_read_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::Client::set_read_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка [2330](#)

```
02330 {  
02331     cli_->set_read_timeout(duration);  
02332 }
```

6.4.3.109 `set_read_timeout()` [2/2]

```
void httpplib::Client::set_read_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле [httpplib.h](#) строка [10400](#)

```
10400 {  
10401     cli_->set_read_timeout(sec, usec);  
10402 }
```

6.4.3.110 `set_socket_options()`

```
void httpplib::Client::set_socket_options (
    SocketOptions socket_options) [inline]
```

См. определение в файле [httpplib.h](#) строка [10392](#)

```
10392 {  
10393     cli_->set_socket_options(std::move(socket_options));  
10394 }
```

6.4.3.111 `set_tcp_nodelay()`

```
void httpplib::Client::set_tcp_nodelay (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка [10390](#)

```
10390 { cli_->set_tcp_nodelay(on); }
```

6.4.3.112 `set_url_encode()`

```
void httpplib::Client::set_url_encode (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 10427

```
10427 { cli_->set_url_encode(on); }
```

6.4.3.113 `set_write_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::Client::set_write_timeout (
    const std::chrono::duration<Rep, Period> & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2336

```
02336 {
```

```
02337   cli_->set_write_timeout(duration);
```

```
02338 }
```

6.4.3.114 `set_write_timeout()` [2/2]

```
void httpplib::Client::set_write_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле [httpplib.h](#) строка 10404

```
10404 {
```

```
10405   cli_->set_write_timeout(sec, usec);
```

```
10406 }
```

6.4.3.115 `socket()`

```
socket_t httpplib::Client::socket () const [inline]
```

См. определение в файле [httpplib.h](#) строка 10370

```
10370 { return cli_->socket(); }
```

6.4.3.116 `stop()`

```
void httpplib::Client::stop () [inline]
```

См. определение в файле [httpplib.h](#) строка 10362

```
10362 { cli_->stop(); }
```

6.4.4 Данные класса

6.4.4.1 `cli_`

```
std::unique_ptr<ClientImpl> httpplib::Client::cli_ [private]
```

См. определение в файле [httpplib.h](#) строка 1946

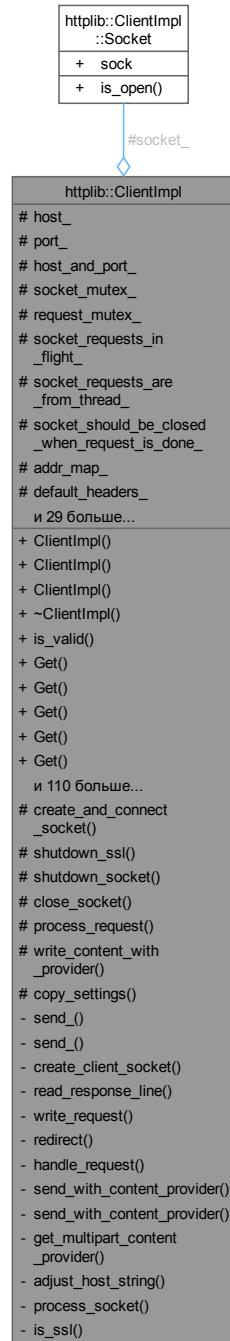
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.5 Класс http://ClientImpl

```
#include <http://ClientImpl.h>
```

Граф связей класса http://ClientImpl:



Классы

- struct [Socket](#)

Открытые члены

- `ClientImpl` (const std::string &`host`)
- `ClientImpl` (const std::string &`host`, int `port`)
- `ClientImpl` (const std::string &`host`, int `port`, const std::string &`client_cert_path`, const std::string &`client_key_path`)
- `virtual ~ClientImpl ()`
- `virtual bool is_valid () const`
- `Result Get` (const std::string &`path`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`)
- `Result Get` (const std::string &`path`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, `ContentReceiver` `content_receiver`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `ContentReceiver` `content_receiver`)
- `Result Get` (const std::string &`path`, `ContentReceiver` `content_receiver`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `ContentReceiver` `content_receiver`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, `ResponseHandler` `response_handler`, `ContentReceiver` `content_receiver`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `ResponseHandler` `response_handler`, `ContentReceiver` `content_receiver`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `ResponseHandler` `response_handler`, `ContentReceiver` `content_receiver`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, const `Headers` &`headers`, `ResponseHandler` `response_handler`, `ContentReceiver` `content_receiver`, `Progress` `progress`)
- `Result Get` (const std::string &`path`, const `Params` &`params`, const `Headers` &`headers`, `Progress` `progress=nullptr`)
- `Result Get` (const std::string &`path`, const `Params` &`params`, const `Headers` &`headers`, `ContentReceiver` `content_receiver`, `Progress` `progress=nullptr`)
- `Result Get` (const std::string &`path`, const `Params` &`params`, const `Headers` &`headers`, `ResponseHandler` `response_handler`, `ContentReceiver` `content_receiver`, `Progress` `progress=nullptr`)
- `Result Head` (const std::string &`path`)
- `Result Head` (const std::string &`path`, const `Headers` &`headers`)
- `Result Post` (const std::string &`path`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`)
- `Result Post` (const std::string &`path`, const char *`body`, size_t `content_length`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, const char *`body`, size_t `content_length`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, const char *`body`, size_t `content_length`, const std::string &`content_type`, `Progress` `progress`)
- `Result Post` (const std::string &`path`, const std::string &`body`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const std::string &`body`, const std::string &`content_type`, `Progress` `progress`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, const std::string &`body`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, const std::string &`body`, const std::string &`content_type`, `Progress` `progress`)
- `Result Post` (const std::string &`path`, const std::string &`body`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, const std::string &`body`, const std::string &`content_type`, `Progress` `progress`)
- `Result Post` (const std::string &`path`, size_t `content_length`, `ContentProvider` `content_provider`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, `ContentProviderWithoutLength` `content_provider`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, size_t `content_length`, `ContentProvider` `content_provider`, const std::string &`content_type`)
- `Result Post` (const std::string &`path`, const `Headers` &`headers`, `ContentProviderWithoutLength` `content_provider`, const std::string &`content_type`)

- `Result Post (const std::string &path, const Params ¶ms)`
- `Result Post (const std::string &path, const Headers &headers, const Params ¶ms)`
- `Result Post (const std::string &path, const Headers &headers, const Params ¶ms, Progress progress)`
- `Result Post (const std::string &path, const MultipartFormDataItems &items)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const std::string &boundary)`
- `Result Post (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const MultipartFormDataProviderItems &provider_items)`
- `Result Put (const std::string &path)`
- `Result Put (const std::string &path, const char *body, size_t content_length, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, ContentProviderWithoutLength content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, size_t content_length, ContentProvider content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Headers &headers, ContentProviderWithoutLength content_provider, const std::string &content_type)`
- `Result Put (const std::string &path, const Params ¶ms)`
- `Result Put (const std::string &path, const Headers &headers, const Params ¶ms)`
- `Result Put (const std::string &path, const Headers &headers, const Params ¶ms, Progress progress)`
- `Result Put (const std::string &path, const MultipartFormDataItems &items)`
- `Result Put (const std::string &path, const Headers &headers, const MultipartFormDataItems &items)`
- `Result Put (const std::string &path, const Headers &headers, const MultipartFormDataItems &items, const std::string &boundary)`
- `Result Put (const std::string &path, const Headers &headers, const char *body, size_t content_length, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type, Progress progress)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type)`
- `Result Put (const std::string &path, const std::string &body, const std::string &content_type, Progress progress)`

- `Result Patch` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type)
- `Result Patch` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Patch` (const std::string &path, size_t content_length, `ContentProvider` content_provider, const std::string &content_type)
- `Result Patch` (const std::string &path, `ContentProviderWithoutLength` content_provider, const std::string &content_type)
- `Result Patch` (const std::string &path, const `Headers` &headers, size_t content_length, `ContentProvider` content_provider, const std::string &content_type)
- `Result Patch` (const std::string &path, const `Headers` &headers, `ContentProviderWithoutLength` content_provider, const std::string &content_type)
- `Result Delete` (const std::string &path)
- `Result Delete` (const std::string &path, const `Headers` &headers)
- `Result Delete` (const std::string &path, const char *body, size_t content_length, const std::string &content_type)
- `Result Delete` (const std::string &path, const char *body, size_t content_length, const std::string &content_type, `Progress` progress)
- `Result Delete` (const std::string &path, const `Headers` &headers, const char *body, size_t content_length, const std::string &content_type)
- `Result Delete` (const std::string &path, const `Headers` &headers, const char *body, size_t content_length, const std::string &content_type, `Progress` progress)
- `Result Delete` (const std::string &path, const std::string &body, const std::string &content_type)
- `Result Delete` (const std::string &path, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Delete` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type)
- `Result Delete` (const std::string &path, const `Headers` &headers, const std::string &body, const std::string &content_type, `Progress` progress)
- `Result Options` (const std::string &path)
- `Result Options` (const std::string &path, const `Headers` &headers)
- `bool send` (`Request` &req, `Response` &res, `Error` &error)
- `Result send` (const `Request` &req)
- `void stop` ()
- `std::string host` () const
- `int port` () const
- `size_t is_socket_open` () const
- `socket_t socket` () const
- `void set_hostname_addr_map` (std::map< std::string, std::string > addr_map)
- `void set_default_headers` (`Headers` headers)
- `void set_header_writer` (std::function< ssize_t(`Stream` &, `Headers` &)> const &writer)
- `void set_address_family` (int family)
- `void set_tcp_nodelay` (bool on)
- `void set_ipv6_v6only` (bool on)
- `void set_socket_options` (`SocketOptions` socket_options)
- `void set_connection_timeout` (time_t sec, time_t usec=0)
- template<class Rep, class Period>
 `void set_connection_timeout` (const std::chrono::duration< Rep, Period > &duration)
- `void set_read_timeout` (time_t sec, time_t usec=0)
- template<class Rep, class Period>
 `void set_read_timeout` (const std::chrono::duration< Rep, Period > &duration)
- `void set_write_timeout` (time_t sec, time_t usec=0)
- template<class Rep, class Period>
 `void set_write_timeout` (const std::chrono::duration< Rep, Period > &duration)
- `void set_max_timeout` (time_t msec)

- template<class Rep, class Period>
void `set_max_timeout` (const std::chrono::duration< Rep, Period > &duration)
- void `set_basic_auth` (const std::string &username, const std::string &password)
- void `set_bearer_token_auth` (const std::string &token)
- void `set_keep_alive` (bool on)
- void `set_follow_location` (bool on)
- void `set_url_encode` (bool on)
- void `set_compress` (bool on)
- void `set_decompress` (bool on)
- void `set_interface` (const std::string &intf)
- void `set_proxy` (const std::string &host, int port)
- void `set_proxy_basic_auth` (const std::string &username, const std::string &password)
- void `set_proxy_bearer_token_auth` (const std::string &token)
- void `set_logger` (Logger logger)

Защищенные члены

- virtual bool `create_and_connect_socket` (Socket &socket, Error &error)
- virtual void `shutdown_ssl` (Socket &socket, bool shutdown_gracefully)
- void `shutdown_socket` (Socket &socket) const
- void `close_socket` (Socket &socket)
- bool `process_request` (Stream &strm, Request &req, Response &res, bool close_connection, Error &error)
- bool `write_content_with_provider` (Stream &strm, const Request &req, Error &error) const
- void `copy_settings` (const ClientImpl &rhs)

Защищенные данные

- const std::string `host_`
- const int `port_`
- const std::string `host_and_port_`
- Socket `socket_`
- std::mutex `socket_mutex_`
- std::recursive_mutex `request_mutex_`
- size_t `socket_requests_in_flight_` = 0
- std::thread::id `socket_requests_are_from_thread_` = std::thread::id()
- bool `socket_should_be_closed_when_request_is_done_` = false
- std::map< std::string, std::string > `addr_map_`
- Headers `default_headers_`
- std::function< ssize_t(Stream &, Headers &)> `header_writer_`
- std::string `client_cert_path_`
- std::string `client_key_path_`
- time_t `connection_timeout_sec_` = 300
- time_t `connection_timeout_usec_` = 0
- time_t `read_timeout_sec_` = 300
- time_t `read_timeout_usec_` = 0
- time_t `write_timeout_sec_` = 5
- time_t `write_timeout_usec_` = 0
- time_t `max_timeout_msec_` = 0
- std::string `basic_auth_username_`
- std::string `basic_auth_password_`
- std::string `bearer_token_auth_token_`
- bool `keep_alive_` = false

- `bool follow_location_ = false`
- `bool url_encode_ = true`
- `int address_family_ = AF_UNSPEC`
- `bool tcp_nodelay_ = false`
- `bool ipv6_v6only_ = false`
- `SocketOptions socket_options_ = nullptr`
- `bool compress_ = false`
- `bool decompress_ = true`
- `std::string interface_`
- `std::string proxy_host_`
- `int proxy_port_ = -1`
- `std::string proxy_basic_auth_username_`
- `std::string proxy_basic_auth_password_`
- `std::string proxy_bearer_token_auth_token_`
- `Logger logger_`

Закрытые члены

- `bool send_ (Request &req, Response &res, Error &error)`
- `Result send_ (Request &&req)`
- `socket_t create_client_socket (Error &error) const`
- `bool read_response_line (Stream &strm, const Request &req, Response &res) const`
- `bool write_request (Stream &strm, Request &req, bool close_connection, Error &error)`
- `bool redirect (Request &req, Response &res, Error &error)`
- `bool handle_request (Stream &strm, Request &req, Response &res, bool close_connection, Error &error)`
- `std::unique_ptr< Response > send_with_content_provider (Request &req, const char *body, size_t content_length, ContentProvider content_provider, ContentProviderWithoutLength content_provider_without_length, const std::string &content_type, Error &error)`
- `Result send_with_content_provider (const std::string &method, const std::string &path, const Headers &headers, const char *body, size_t content_length, ContentProvider content_provider, ContentProviderWithoutLength content_provider_without_length, const std::string &content_type, Progress progress)`
- `ContentProviderWithoutLength get_multipart_content_provider (const std::string &boundary, const MultipartFormDataItems &items, const MultipartFormDataProviderItems &provider_items) const`
- `std::string adjust_host_string (const std::string &host) const`
- `virtual bool process_socket (const Socket &socket, std::chrono::time_point< std::chrono::steady_clock > start_time, std::function< bool(Stream &strm)> callback)`
- `virtual bool is_ssl () const`

6.5.1 Подробное описание

См. определение в файле `httpplib.h` строка 1224

6.5.2 Конструктор(ы)

6.5.2.1 ClientImpl() [1/3]

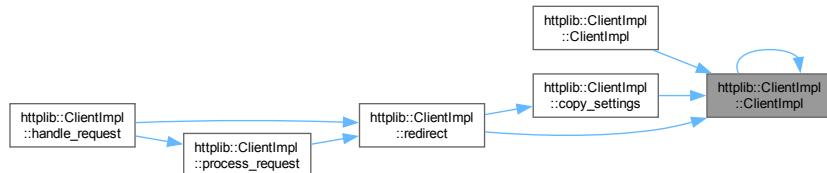
```
httplib::ClientImpl::ClientImpl (
    const std::string & host) [inline], [explicit]
```

См. определение в файле [httplib.h](#) строка [7447](#)
07448 : [ClientImpl\(host, 80, std::string\(\), std::string\(\)\) {}](#)

Граф вызовов:



Граф вызова функции:

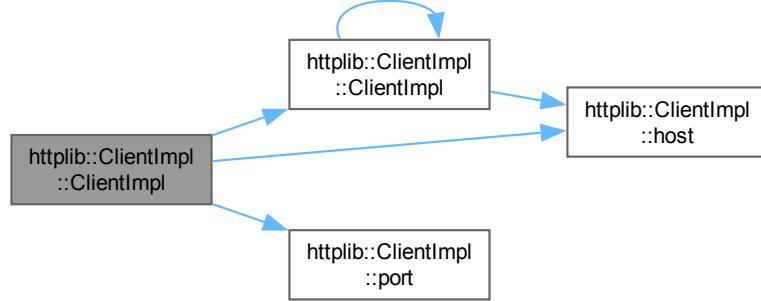


6.5.2.2 ClientImpl() [2/3]

```
httplib::ClientImpl::ClientImpl (
    const std::string & host,
    int port) [inline], [explicit]
```

См. определение в файле [httplib.h](#) строка [7450](#)
07451 : [ClientImpl\(host, port, std::string\(\), std::string\(\)\) {}](#)

Граф вызовов:



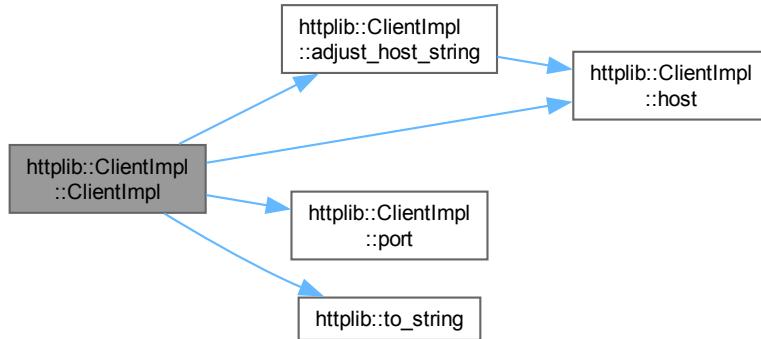
6.5.2.3 ClientImpl() [3/3]

```
http://ClientImpl::ClientImpl (
    const std::string & host,
    int port,
    const std::string & client_cert_path,
    const std::string & client_key_path) [inline], [explicit]
```

См. определение в файле [http://Client.h](#) строка 7453

```
07456 : host_(detail::escape_abstract_namespace_unix_domain(host)), port_(port),
07457 host_and_port_(adjust_host_string(host_) + ":" + std::to_string(port)),
07458 client_cert_path_(client_cert_path), client_key_path_(client_key_path) {}
```

Граф вызовов:



6.5.2.4 ~ClientImpl()

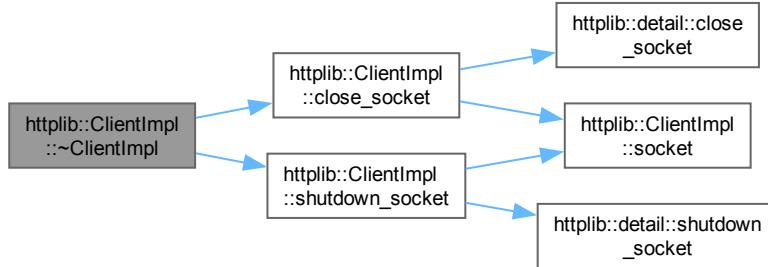
httplib::ClientImpl::~ClientImpl () [inline], [virtual]

См. определение в файле [httplib.h](#) строка 7460

```

07460 {
07461 // Wait until all the requests in flight are handled.
07462 size_t retry_count = 10;
07463 while (retry_count-- > 0) {
07464 {
07465     std::lock_guard<std::mutex> guard(socket_mutex_);
07466     if (socket_requests_in_flight_ == 0) { break; }
07467 }
07468 std::this_thread::sleep_for(std::chrono::milliseconds{1});
07469 }
07470
07471 std::lock_guard<std::mutex> guard(socket_mutex_);
07472 shutdown_socket(socket_);
07473 close_socket(socket_);
07474 }
```

Граф вызовов:



6.5.3 Методы

6.5.3.1 adjust_host_string()

std::string httplib::ClientImpl::adjust_host_string (
const std::string & host) const [inline], [private]

См. определение в файле [httplib.h](#) строка 8148

```

08148 {
08149 if (host.find(':') != std::string::npos) { return "[" + host + "]"; }
08150 return host;
08151 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.2 close_socket()

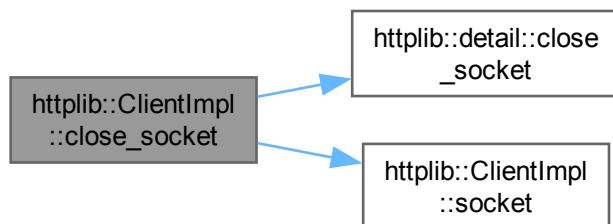
```
void httplib::ClientImpl::close_socket (
    Socket & socket) [inline], [protected]
```

См. определение в файле [httplib.h](#) строка 7568

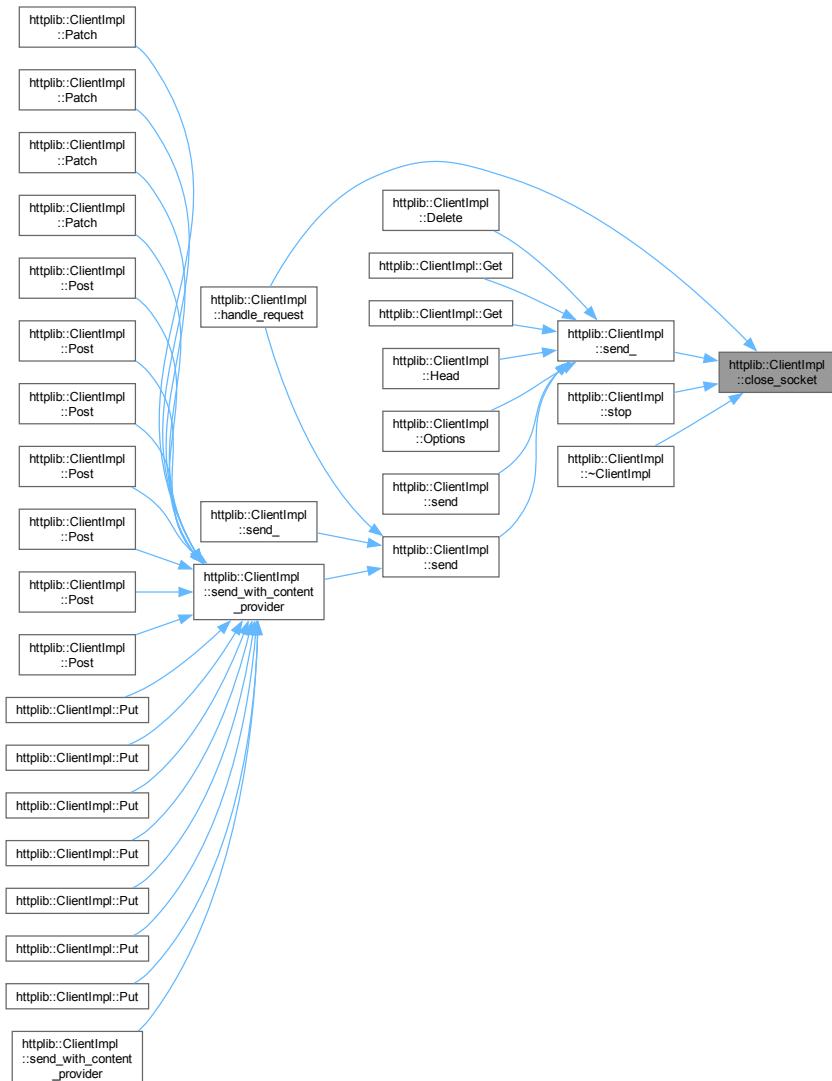
```

07568     {
07569     // If there are requests in flight in another thread, usually closing
07570     // the socket will be fine and they will simply receive an error when
07571     // using the closed socket, but it is still a bug since rarely the OS
07572     // may reassign the socket id to be used for a new socket, and then
07573     // suddenly they will be operating on a live socket that is different
07574     // than the one they intended!
07575     assert(socket_requests_in_flight_ == 0 ||
07576         socket_requests_are_from_thread_ == std::this_thread::get_id());
07577
07578     // It is also a bug if this happens while SSL is still active
07579 #ifndef CPPHTTPPLIB_OPENSSL_SUPPORT
07580     assert(socket.ssl == nullptr);
07581 #endif
07582     if (socket.sock == INVALID_SOCKET) { return; }
07583     detail::close_socket(socket.sock);
07584     socket.sock = INVALID_SOCKET;
07585 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.3 copy_settings()

```
void http://ClientImpl::copy_settings(
    const ClientImpl & rhs) [inline], [protected]
```

См. определение в файле [http://ClientImpl.h](#) строка 7478

```
07478 {
07479     client_cert_path_ = rhs.client_cert_path_;
07480     client_key_path_ = rhs.client_key_path_;
07481     connection_timeout_sec_ = rhs.connection_timeout_sec_;
07482     read_timeout_sec_ = rhs.read_timeout_sec_;
07483     read_timeout_usec_ = rhs.read_timeout_usec_;
07484     write_timeout_sec_ = rhs.write_timeout_sec_;
07485     write_timeout_usec_ = rhs.write_timeout_usec_;
07486     max_timeout_msec_ = rhs.max_timeout_msec_;
07487     basic_auth_username_ = rhs.basic_auth_username_;
07488     basic_auth_password_ = rhs.basic_auth_password_;
07489     bearer_token_auth_token_ = rhs.bearer_token_auth_token_;
07490 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
```

```

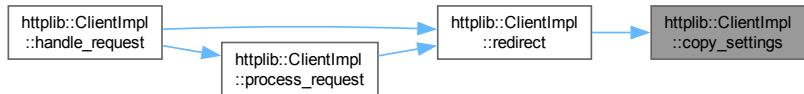
07491 digest_auth_username_ = rhs.digest_auth_username_;
07492 digest_auth_password_ = rhs.digest_auth_password_;
07493 #endif
07494 keep_alive_ = rhs.keep_alive_;
07495 follow_location_ = rhs.follow_location_;
07496 url_encode_ = rhs.url_encode_;
07497 address_family_ = rhs.address_family_;
07498 tcp_nodelay_ = rhs.tcp_nodelay_;
07499 ipv6_v6only_ = rhs.ipv6_v6only_;
07500 socket_options_ = rhs.socket_options_;
07501 compress_ = rhs.compress_;
07502 decompress_ = rhs.decompress_;
07503 interface_ = rhs.interface_;
07504 proxy_host_ = rhs.proxy_host_;
07505 proxy_port_ = rhs.proxy_port_;
07506 proxy_basic_auth_username_ = rhs.proxy_basic_auth_username_;
07507 proxy_basic_auth_password_ = rhs.proxy_basic_auth_password_;
07508 proxy_bearer_token_auth_token_ = rhs.proxy_bearer_token_auth_token_;
07509 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
07510 proxy_digest_auth_username_ = rhs.proxy_digest_auth_username_;
07511 proxy_digest_auth_password_ = rhs.proxy_digest_auth_password_;
07512 #endif
07513 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
07514 ca_cert_file_path_ = rhs.ca_cert_file_path_;
07515 ca_cert_dir_path_ = rhs.ca_cert_dir_path_;
07516 ca_cert_store_ = rhs.ca_cert_store_;
07517 #endif
07518 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
07519 server_certificate_verification_ = rhs.server_certificate_verification_;
07520 server_hostname_verification_ = rhs.server_hostname_verification_;
07521 server_certificate_verifier_ = rhs.server_certificate_verifier_;
07522 #endif
07523 logger_ = rhs.logger_;
07524 }

```

Граф вызовов:



Граф вызова функции:



6.5.3.4 create_and_connect_socket()

```

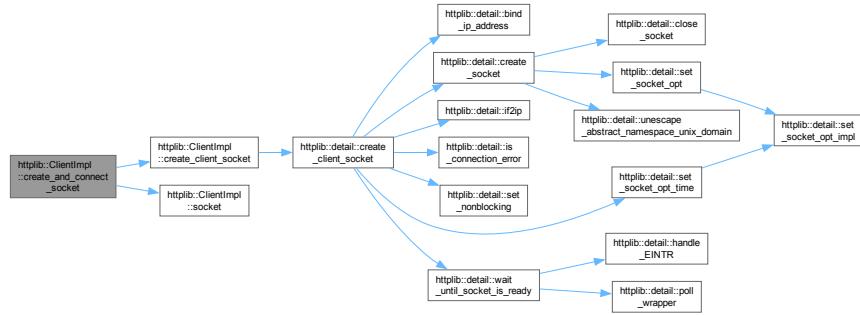
bool httplib::ClientImpl::create_and_connect_socket (
    Socket & socket,
    Error & error) [inline], [protected], [virtual]

```

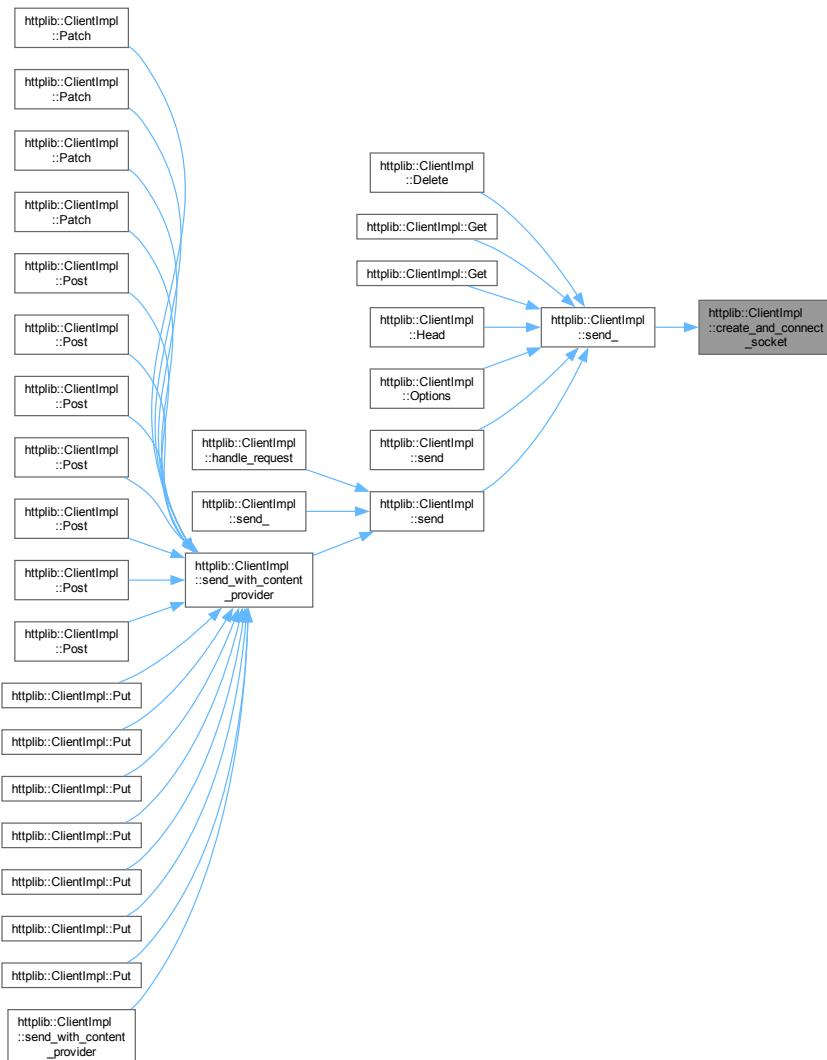
См. определение в файле [httplib.h](#) строка 7547

```
07548 {  
07549     auto sock = create_client_socket(error);  
07550     if (sock == INVALID_SOCKET) { return false; }  
07551     socket.sock = sock;  
07552     return true;  
07553 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.5 create_client_socket()

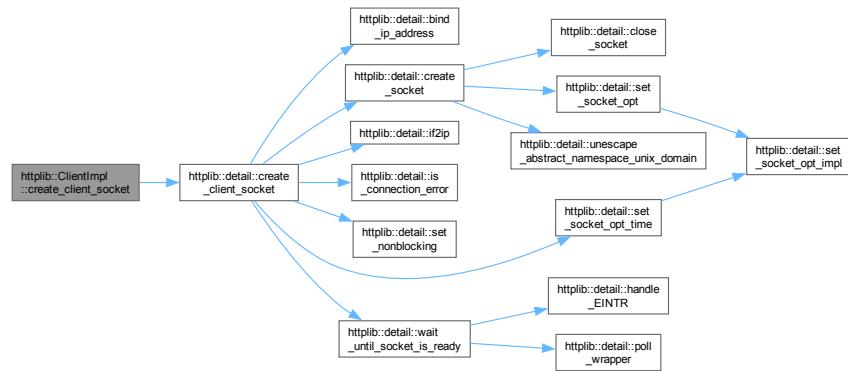
```
socket_t httplib::ClientImpl::create_client_socket (
    Error & error) const [inline], [private]
```

См. определение в файле [httplib.h](#) строка 7526

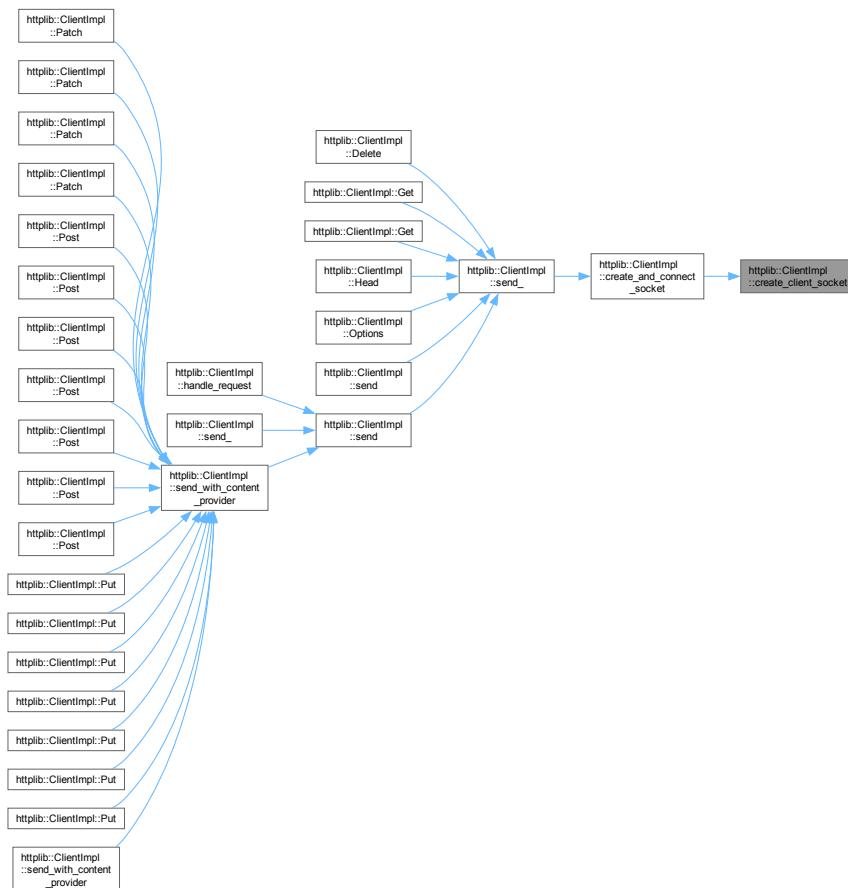
```
07526 if (!proxy_host_.empty() && proxy_port_ != -1) {
07527     return detail::create_client_socket(
07528         proxy_host_, std::string(), proxy_port_, address_family_, tcp_nodelay_,
07529         ipv6_v6only_, socket_options_, connection_timeout_sec_,
07530         connection_timeout_usec_, read_timeout_sec_, read_timeout_usec_,
07531         write_timeout_sec_, write_timeout_usec_, interface_, error);
07532 }
07533
07534 // Check is custom IP specified for host_
07535 std::string ip;
07536 auto it = addr_map_.find(host_);
07537 if (it != addr_map_.end()) { ip = it->second; }
07538 }
```

```
07540     return detail::create_client_socket(
07541         host_, ip, port_, address_family_, tcp_nodelay_, ipv6_v6only_,
07542         socket_options_, connection_timeout_sec_, connection_timeout_usec_,
07543         read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
07544         write_timeout_usec_, interface_, error);
07545 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.6 `Delete()` [1/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path) [inline]
```

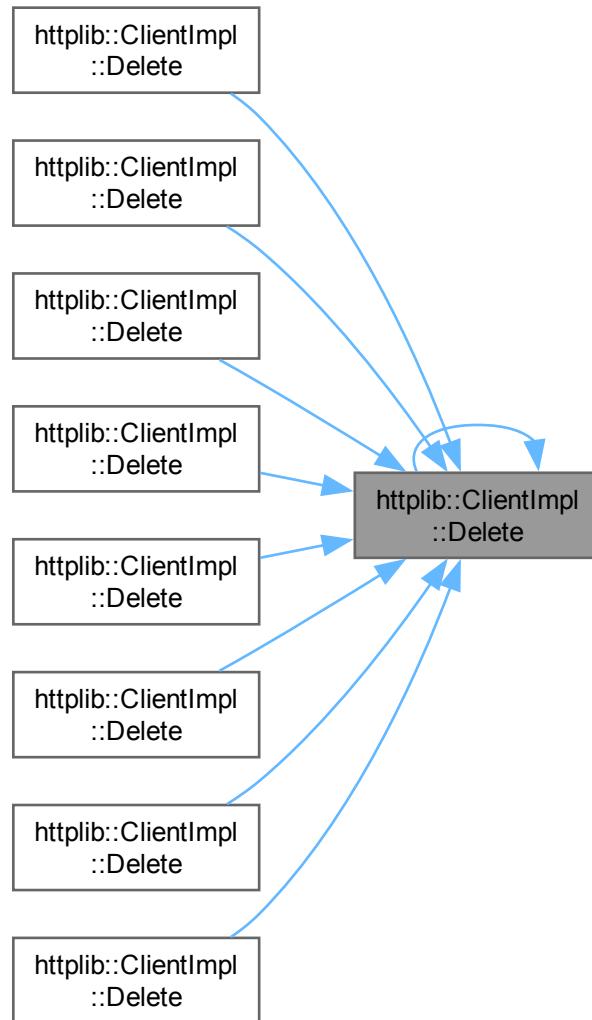
См. определение в файле [httpplib.h](#) строка [8815](#)

```
08815     {
08816     return Delete(path, Headers(), std::string(), std::string());
08817 }
```

Граф вызовов:



Граф вызова функции:



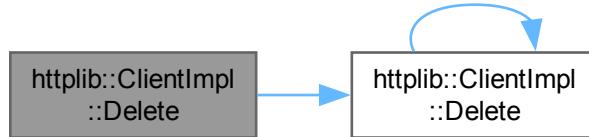
6.5.3.7 Delete() [2/10]

```
Result httplib::ClientImpl::Delete (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httplib.h` строка 8824

```
08826 {  
08827     return Delete(path, Headers(), body, content_length, content_type);  
08828 }
```

Граф вызовов:



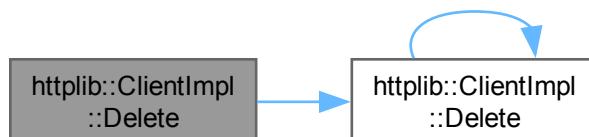
6.5.3.8 Delete() [3/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8830

```
08833 {
08834     return Delete(path, Headers(), body, content_length, content_type, progress);
08835 }
```

Граф вызовов:



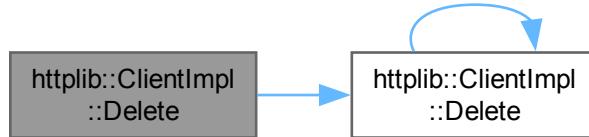
6.5.3.9 Delete() [4/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 8819

```
08820 {
08821     return Delete(path, headers, std::string(), std::string());
08822 }
```

Граф вызовов:



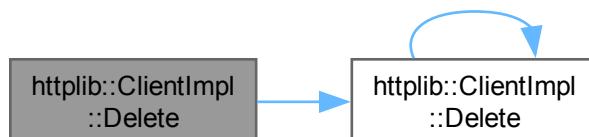
6.5.3.10 Delete() [5/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле [httpplib.h](#) строка 8837

```
08840 {
08841     return Delete(path, headers, body, content_length, content_type, nullptr);
08842 }
```

Граф вызовов:



6.5.3.11 Delete() [6/10]

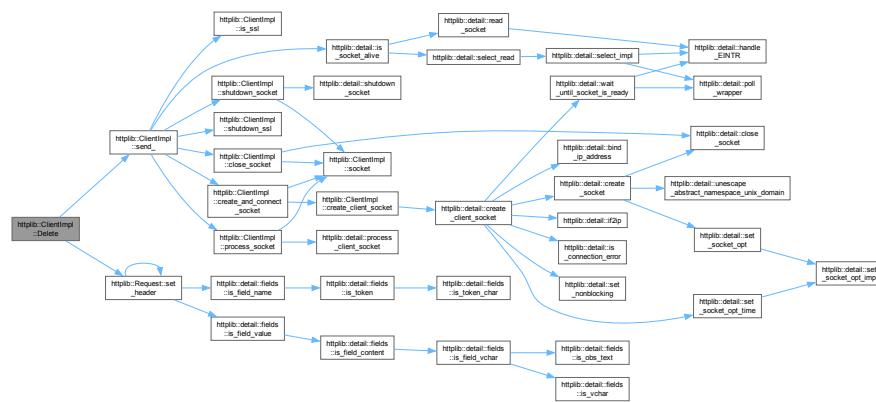
```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8844

```

08848     {
08849     Request req;
08850     req.method = "DELETE";
08851     req.headers = headers;
08852     req.path = path;
08853     req.progress = progress;
08854     if (max_timeout_msec > 0) {
08855         req.start_time_ = std::chrono::steady_clock::now();
08856     }
08857
08858     if (!content_type.empty()) { req.set_header("Content-Type", content_type); }
08859     req.body.assign(body, content_length);
08860
08861     return send_(std::move(req));
08862 }
```

Граф вызовов:



6.5.3.12 Delete() [7/10]

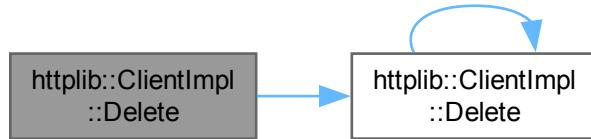
```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 8878

```

08881     {
08882     return Delete(path, headers, body.data(), body.size(), content_type);
08883 }
```

Граф вызовов:



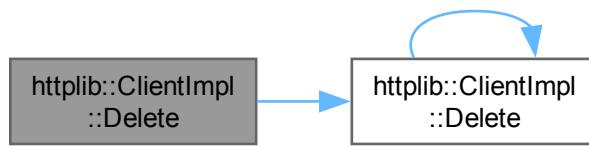
6.5.3.13 `Delete()` [8/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8885

```
08889 {
08890     return Delete(path, headers, body.data(), body.size(), content_type,
08891                 progress);
08892 }
```

Граф вызовов:

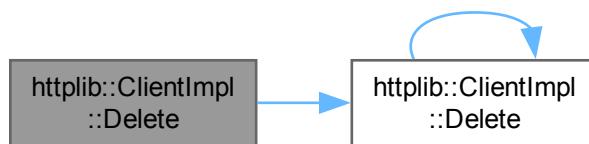
6.5.3.14 `Delete()` [9/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 8864

```
08866 {
08867     return Delete(path, Headers(), body.data(), body.size(), content_type);
08868 }
```

Граф вызовов:



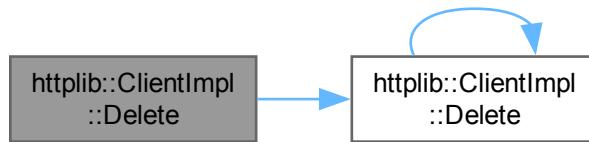
6.5.3.15 `Delete()` [10/10]

```
Result httpplib::ClientImpl::Delete (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8870

```
08873     {
08874     return Delete(path, Headers(), body.data(), body.size(), content_type,
08875                 progress);
08876 }
```

Граф вызовов:

6.5.3.16 `Get()` [1/15]

```
Result httpplib::ClientImpl::Get (
    const std::string & path) [inline]
```

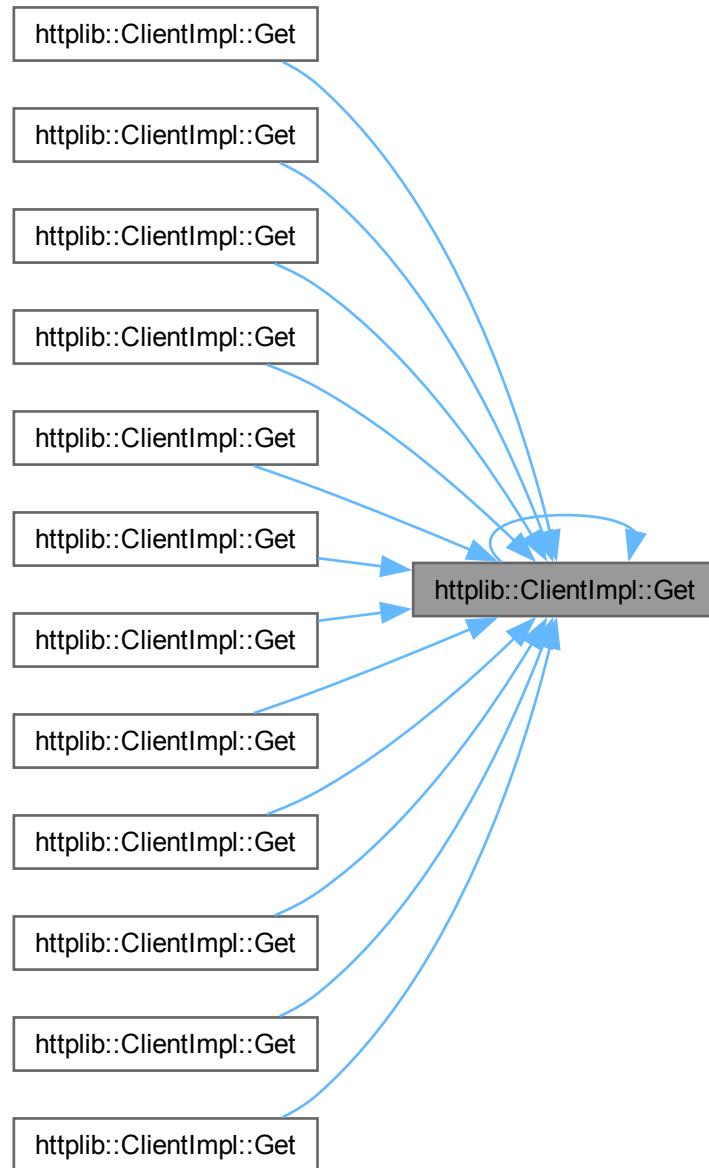
См. определение в файле `httpplib.h` строка 8299

```
08299     {
08300     return Get(path, Headers(), Progress());
08301 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.17 Get() [2/15]

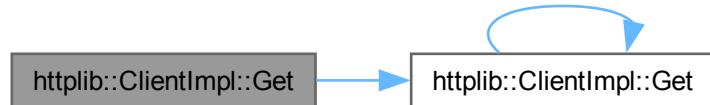
```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле [httplib.h](#) строка 8307

```
08307
08308 return Get(path, headers, Progress());
```

```
08309 }
```

Граф вызовов:



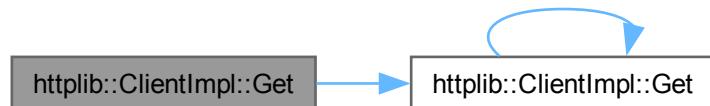
6.5.3.18 Get() [3/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httplib.h` строка 8337

```
08338 {  
08339     return Get(path, headers, nullptr, std::move(content_receiver), nullptr);  
08340 }
```

Граф вызовов:



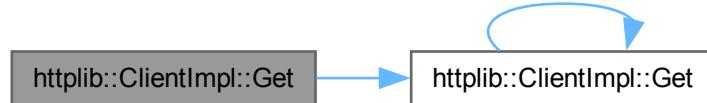
6.5.3.19 Get() [4/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httplib.h` строка 8342

```
08344 {  
08345     return Get(path, headers, nullptr, std::move(content_receiver),  
08346             std::move(progress));  
08347 }
```

Граф вызовов:



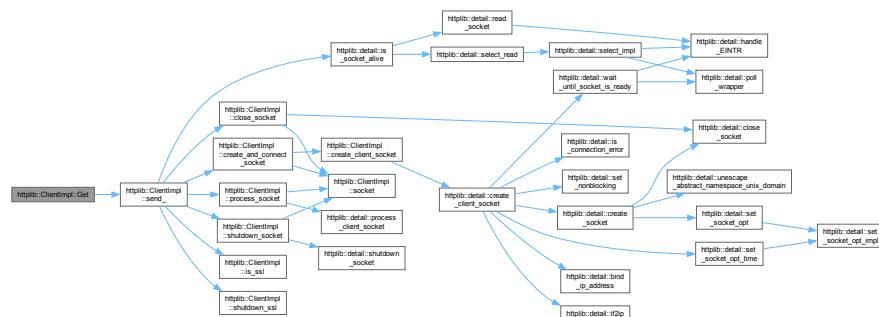
6.5.3.20 Get() [5/15]

```
Result httpplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8311

```
08312 {
08313     Request req;
08314     req.method = "GET";
08315     req.path = path;
08316     req.headers = headers;
08317     req.progress = std::move(progress);
08318     if (max_timeout_msec > 0) {
08319         req.start_time_ = std::chrono::steady_clock::now();
08320     }
08321
08322     return send_(std::move(req));
08323 }
```

Граф вызовов:



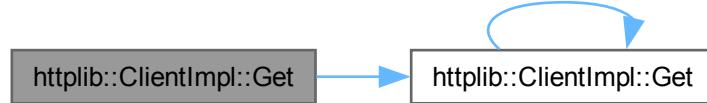
6.5.3.21 Get() [6/15]

```
Result httpplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле `httpplib.h` строка 8356

```
08358     return Get(path, headers, std::move(response_handler),
08359             std::move(content_receiver), nullptr);
08360 }
08361 }
```

Граф вызовов:



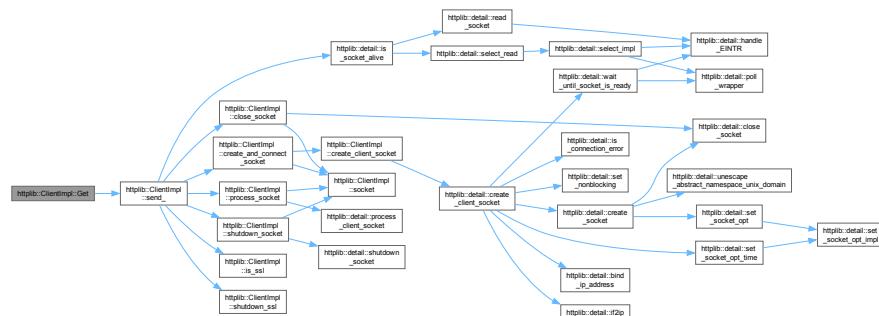
6.5.3.22 Get() [7/15]

```
Result httpplib::ClientImpl::Get (
    const std::string & path,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле `httpplib.h` строка 8371

```
08374     {
08375         Request req;
08376         req.method = "GET";
08377         req.path = path;
08378         req.headers = headers;
08379         req.response_handler = std::move(response_handler);
08380         req.content_receiver =
08381             [content_receiver](const char *data, size_t data_length,
08382                 uint64_t /*offset*/, uint64_t /*total_length*/) {
08383                 return content_receiver(data, data_length);
08384             };
08385         req.progress = std::move(progress);
08386         if (max_timeout_msec_ > 0) {
08387             req.start_time_ = std::chrono::steady_clock::now();
08388         }
08389         return send_(std::move(req));
08390     }
08391 }
```

Граф вызовов:



6.5.3.23 Get() [8/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    ContentReceiver content_receiver,
    Progress progress = nullptr) [inline]
```

См. определение в файле [httplib.h](#) строка 8401

```
08404     {
08405     return Get(path, params, headers, nullptr, std::move(content_receiver),
08406                 std::move(progress));
08407 }
```

Граф вызовов:



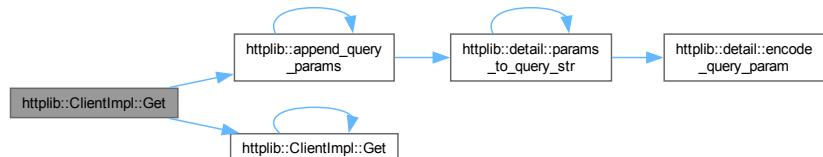
6.5.3.24 Get() [9/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    Progress progress = nullptr) [inline]
```

См. определение в файле [httplib.h](#) строка 8393

```
08394     if (params.empty()) { return Get(path, headers); }
08395     {
08396     std::string path_with_query = append_query_params(path, params);
08397     return Get(path_with_query, headers, std::move(progress));
08398     }
08399 }
```

Граф вызовов:



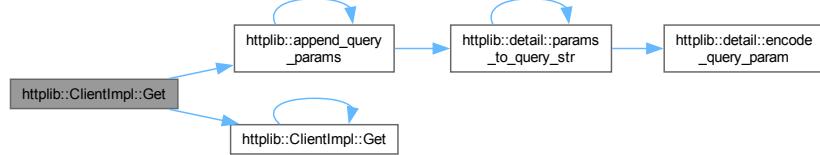
6.5.3.25 Get() [10/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    const Params & params,
    const Headers & headers,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress = nullptr) [inline]
```

См. определение в файле [httplib.h](#) строка 8409

```
08413     if (params.empty()) {
08414         return Get(path, headers, std::move(response_handler),
08415                     std::move(content_receiver), std::move(progress));
08416     }
08417
08418
08419     std::string path_with_query = append_query_params(path, params);
08420     return Get(path_with_query, headers, std::move(response_handler),
08421                 std::move(content_receiver), std::move(progress));
08422 }
```

Граф вызовов:



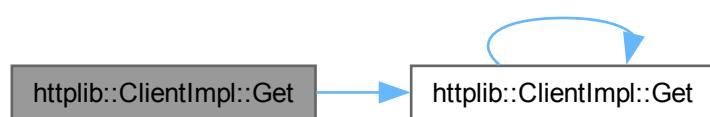
6.5.3.26 Get() [11/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле [httplib.h](#) строка 8325

```
08326     {
08327     return Get(path, Headers(), nullptr, std::move(content_receiver), nullptr);
08328 }
```

Граф вызовов:



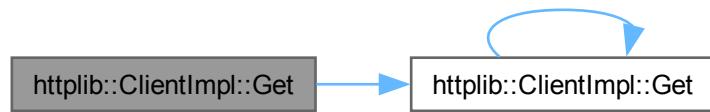
6.5.3.27 Get() [12/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8330

```
08332 {
08333     return Get(path, Headers(), nullptr, std::move(content_receiver),
08334         std::move(progress));
08335 }
```

Граф вызовов:



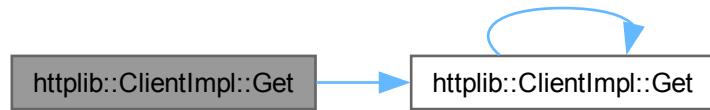
6.5.3.28 Get() [13/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8303

```
08303 {
08304     return Get(path, Headers(), std::move(progress));
08305 }
```

Граф вызовов:



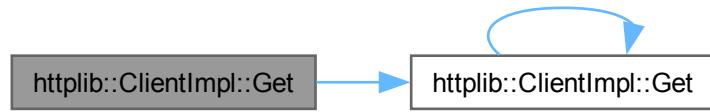
6.5.3.29 Get() [14/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    ResponseHandler response_handler,
    ContentReceiver content_receiver) [inline]
```

См. определение в файле [httplib.h](#) строка 8349

```
08351 {
08352     return Get(path, Headers(), std::move(response_handler),
08353             std::move(content_receiver), nullptr);
08354 }
```

Граф вызовов:



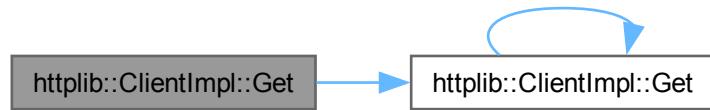
6.5.3.30 Get() [15/15]

```
Result httplib::ClientImpl::Get (
    const std::string & path,
    ResponseHandler response_handler,
    ContentReceiver content_receiver,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8363

```
08366 {
08367     return Get(path, Headers(), std::move(response_handler),
08368             std::move(content_receiver), std::move(progress));
08369 }
```

Граф вызовов:



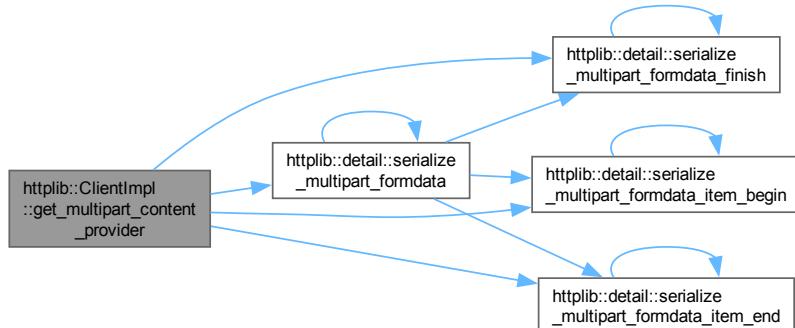
6.5.3.31 `get_multipart_content_provider()`

```
ContentProviderWithoutLength httpplib::ClientImpl::get_multipart_content_provider (
    const std::string & boundary,
    const MultipartFormDataItems & items,
    const MultipartFormDataProviderItems & provider_items) const [inline], [private]
```

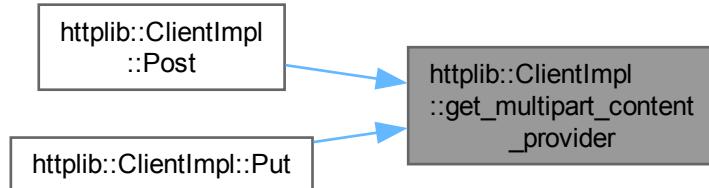
См. определение в файле `httpplib.h` строка 8244

```
08246     size_t cur_item = 0;
08247     size_t cur_start = 0;
08248     // cur_item and cur_start are copied to within the std::function and maintain
08249     // state between successive calls
08250     return [&, cur_item, cur_start](size_t offset,
08251         DataSink & sink) mutable -> bool {
08252         if (!offset && !items.empty()) {
08253             sink.os << detail::serialize_multipart_formdata(items, boundary, false);
08254             return true;
08255         } else if (cur_item < provider_items.size()) {
08256             if (!cur_start) {
08257                 const auto & begin = detail::serialize_multipart_formdata_item_begin(
08258                     provider_items[cur_item], boundary);
08259                 offset += begin.size();
08260                 cur_start = offset;
08261                 sink.os << begin;
08262             }
08263             DataSink cur_sink;
08264             auto has_data = true;
08265             cur_sink.write = sink.write;
08266             cur_sink.done = [&]() { has_data = false; };
08267             if (!provider_items[cur_item].provider(offset - cur_start, cur_sink)) {
08268                 return false;
08269             }
08270             if (!has_data) {
08271                 sink.os << detail::serialize_multipart_formdata_item_end();
08272                 cur_item++;
08273                 cur_start = 0;
08274             }
08275             return true;
08276         } else {
08277             sink.os << detail::serialize_multipart_formdata_finish(boundary);
08278             sink.done();
08279             return true;
08280         }
08281     };
08282 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.32 handle_request()

```
bool http://ClientImpl::handle_request (
    Stream & strm,
    Request & req,
    Response & res,
    bool close_connection,
    Error & error) [inline], [private]
```

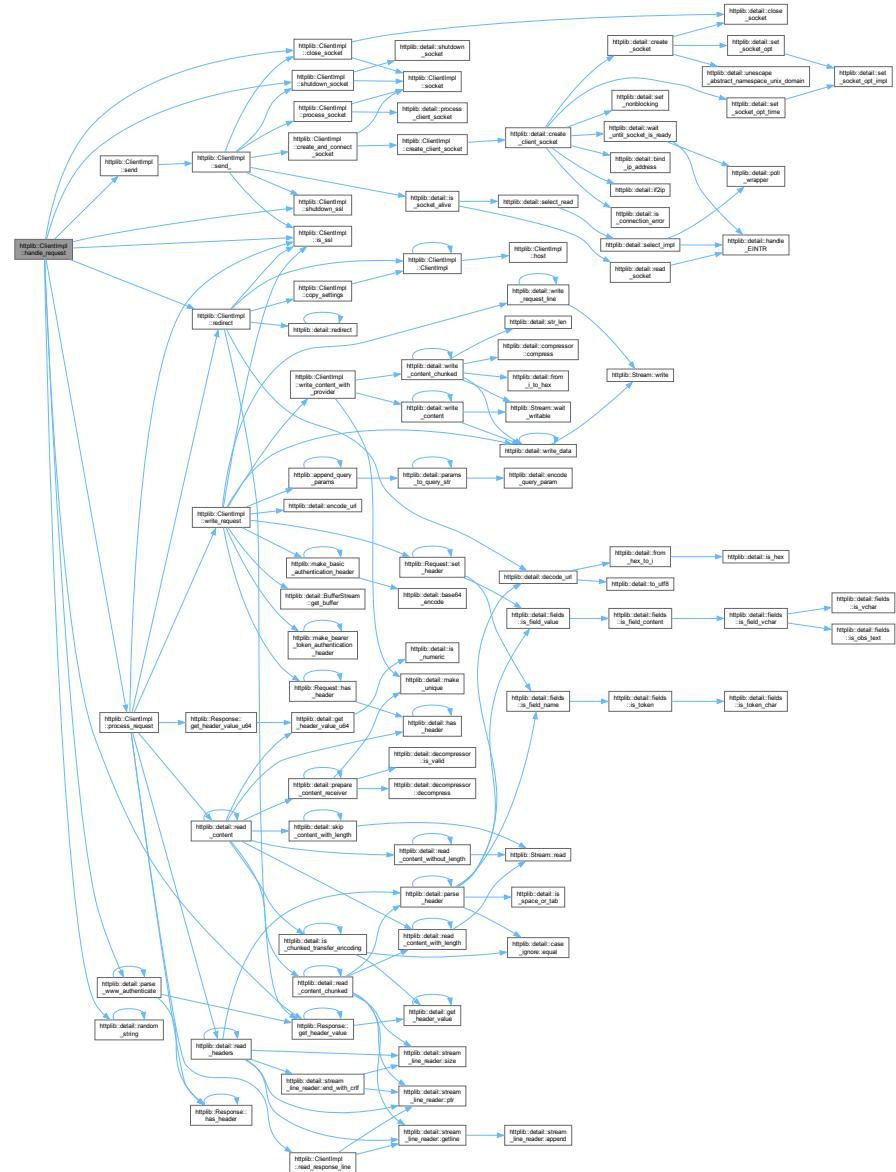
См. определение в файле [http://ClientImpl.h](#) строка 7744

```

07746     if (req.path.empty()) {
07747         error = Error::Connection;
07748         return false;
07749     }
07750
07751     auto req_save = req;
07752     bool ret;
07753
07754     if (!lis_ssl() && !proxy_host_.empty() && proxy_port_ != -1) {
07755         auto req2 = req;
07756         req2.path = "http://" + host_and_port_ + req.path;
07757         ret = process_request(strm, req2, res, close_connection, error);
07758         req = req2;
07759         req.path = req_save.path;
07760     } else {
07761         ret = process_request(strm, req, res, close_connection, error);
07762     }
07763
07764     if (!ret) { return false; }
07765
07766     if (res.get_header_value("Connection") == "close" ||
07767         (res.version == "HTTP/1.0" && res.reason != "Connection established")) {
07768         // TODO this requires a not-entirely-obvious chain of calls to be correct
07769         // for this to be safe.
07770
07771         // This is safe to call because handle_request is only called by send_
07772         // which locks the request mutex during the process. It would be a bug
07773         // to call it from a different thread since it's a thread-safety issue
07774         // to do these things to the socket if another thread is using the socket.
07775         std::lock_guard<std::mutex> guard(socket_mutex_);
07776         shutdown_ssl(socket_, true);
07777         shutdown_socket(socket_);
07778         close_socket(socket_);
07779     }
07780
07781
07782     if (300 < res.status && res.status < 400 && follow_location_) {
07783         req = req_save;
07784         ret = redirect(req, res, error);
07785     }
07786
07787 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
  
```

```
07789 if ((res.status == StatusCode::Unauthorized_401 ||
07790     res.status == StatusCode::ProxyAuthenticationRequired_407) &&
07791     req.authorization_count_ < 5) {
07792     auto is_proxy = res.status == StatusCode::ProxyAuthenticationRequired_407;
07793     const auto &username =
07794         is_proxy ? proxy_digest_auth_username_ : digest_auth_username_;
07795     const auto &password =
07796         is_proxy ? proxy_digest_auth_password_ : digest_auth_password_;
07797
07798     if (!username.empty() && !password.empty()) {
07799         std::map<std::string, std::string> auth;
07800         if (detail::parse_www_authenticate(res, auth, is_proxy)) {
07801             Request new_req = req;
07802             new_req.authorization_count_ += 1;
07803             new_req.headers.erase(is_proxy ? "Proxy-Authorization"
07804                                     : "Authorization");
07805             new_req.headers.insert(detail::make_digest_authentication_header(
07806                 req, auth, new_req.authorization_count_, detail::random_string(10),
07807                 username, password, is_proxy));
07808
07809             Response new_res;
07810
07811             ret = send(new_req, new_res, error);
07812             if (ret) { res = new_res; }
07813         }
07814     }
07815 }
07816 #endif
07817
07818 return ret;
07819 }
```

Граф вызовов:



6.5.3.33 Head() [1/2]

```
Result httpplib::ClientImpl::Head (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 8424

```
08424     return Head(path, Headers());
08425 }
08426 }
```

Граф вызовов:



Граф вызова функции:



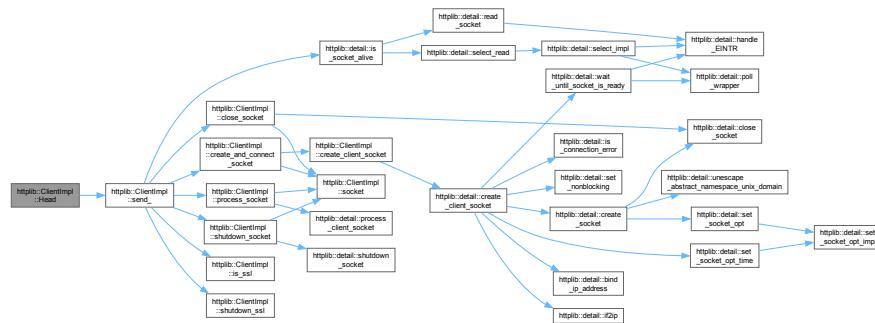
6.5.3.34 Head() [2/2]

```
Result httplib::ClientImpl::Head (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле [httplib.h](#) строка 8428

```
08429
08430 Request req;
08431 req.method = "HEAD";
08432 req.headers = headers;
08433 req.path = path;
08434 if (max_timeout_msec_ > 0) {
08435     req.start_time_ = std::chrono::steady_clock::now();
08436 }
08437 return send_(std::move(req));
08438 }
```

Граф вызовов:

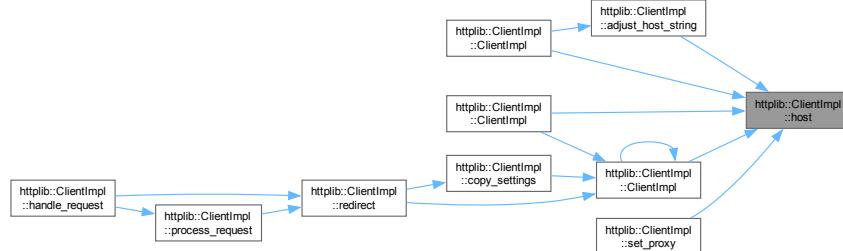


6.5.3.35 host()

`std::string http://ClientImpl::host () const [inline]`

См. определение в файле [http://ClientImpl.h](#) строка 8934
08934 { `return host_;` }

Граф вызова функции:



6.5.3.36 is_socket_open()

`size_t http://ClientImpl::is_socket_open () const [inline]`

См. определение в файле [http://ClientImpl.h](#) строка 8938

```

08938     {
08939     std::lock_guard<std::mutex> guard(socket_mutex_);
08940     return socket_.is_open();
08941   }
  
```

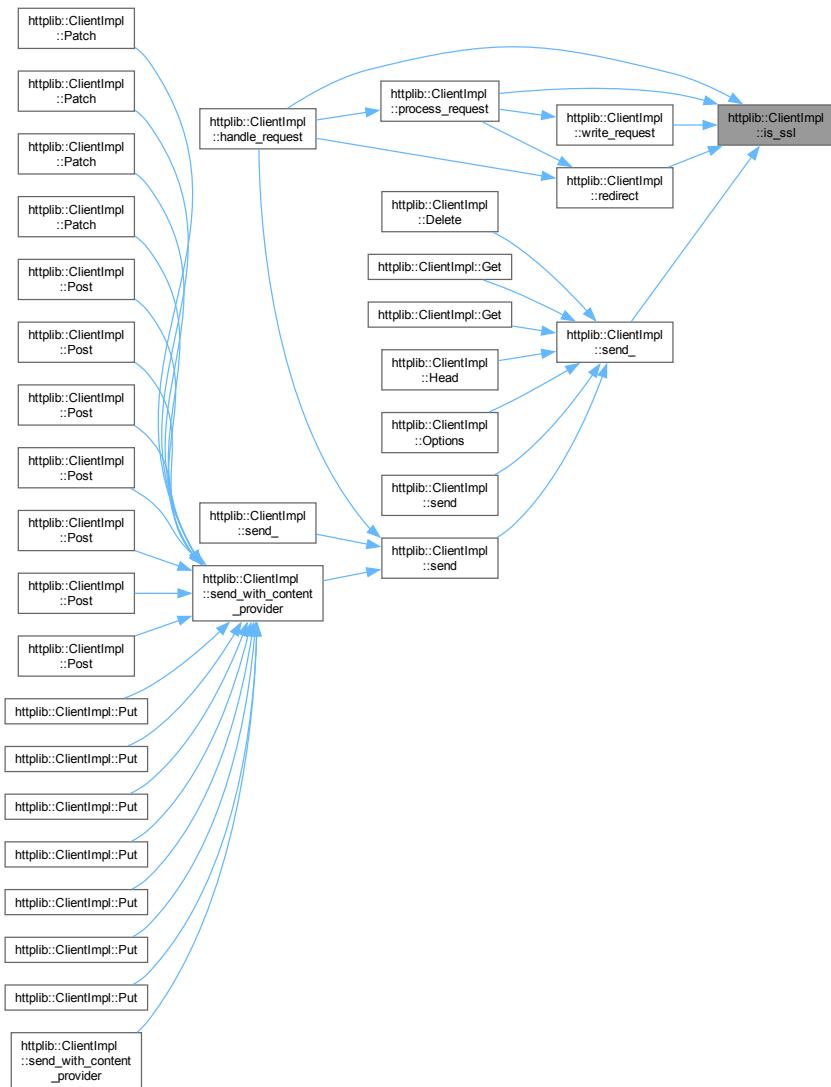
6.5.3.37 is_ssl()

```
bool httplib::ClientImpl::is_ssl () const [inline], [private], [virtual]
```

См. определение в файле [httplib.h](#) строка 8297

```
08297 { return false; }
```

Граф вызова функции:



6.5.3.38 is_valid()

```
bool httplib::ClientImpl::is_valid () const [inline], [virtual]
```

См. определение в файле [httplib.h](#) строка 7476

```
07476 { return true; }
```

6.5.3.39 `Options()` [1/2]

```
Result httpplib::ClientImpl::Options (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 8894

```
08894 {  
08895     return Options(path, Headers());  
08896 }
```

Граф вызовов:



Граф вызова функции:

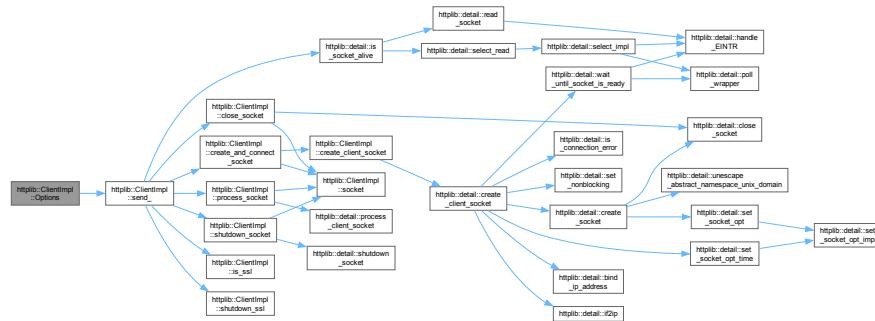
6.5.3.40 `Options()` [2/2]

```
Result httpplib::ClientImpl::Options (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле `httpplib.h` строка 8898

```
08899 {  
08900     Request req;  
08901     req.method = "OPTIONS";  
08902     req.headers = headers;  
08903     req.path = path;  
08904     if (max_timeout_msec_ > 0) {  
08905         req.start_time_ = std::chrono::steady_clock::now();  
08906     }  
08907     return send_(std::move(req));  
08908 }  
08909 }
```

Граф вызовов:



6.5.3.41 Patch() [1/13]

```
Result httpplib::ClientImpl::Patch (
    const std::string & path) [inline]
```

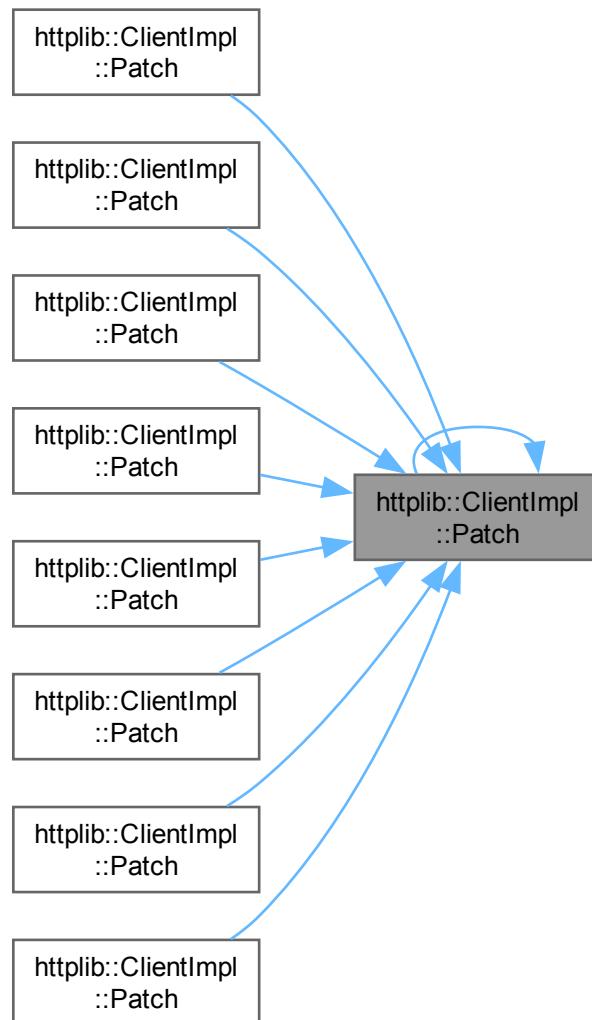
См. определение в файле `httpplib.h` строка 8725

```
08725 {
08726     return Patch(path, std::string(), std::string());
08727 }
```

Граф вызовов:



Граф вызова функции:



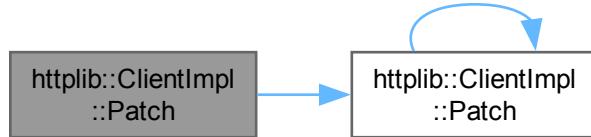
6.5.3.42 Patch() [2/13]

```
Result httplib::ClientImpl::Patch (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8729

```
08731     {
08732     return Patch(path, Headers(), body, content_length, content_type);
08733 }
```

Граф вызовов:



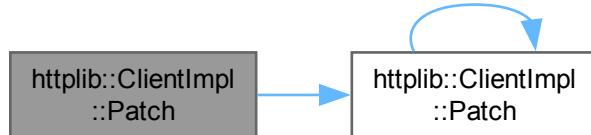
6.5.3.43 Patch() [3/13]

```
Result httpplib::ClientImpl::Patch (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле [httpplib.h](#) строка 8735

```
08738 {
08739     return Patch(path, Headers(), body, content_length, content_type, progress);
08740 }
```

Граф вызовов:



6.5.3.44 Patch() [4/13]

```
Result httpplib::ClientImpl::Patch (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

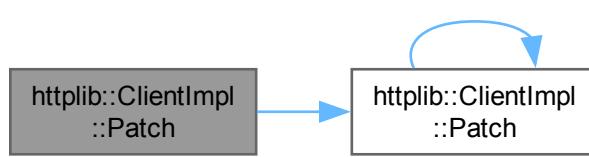
См. определение в файле [httpplib.h](#) строка 8742

```

08744 {
08745     return Patch(path, headers, body, content_length, content_type, nullptr);
08746 }

```

Граф вызовов:



6.5.3.45 Patch() [5/13]

```

Result httpplib::ClientImpl::Patch (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type,
    Progress progress) [inline]

```

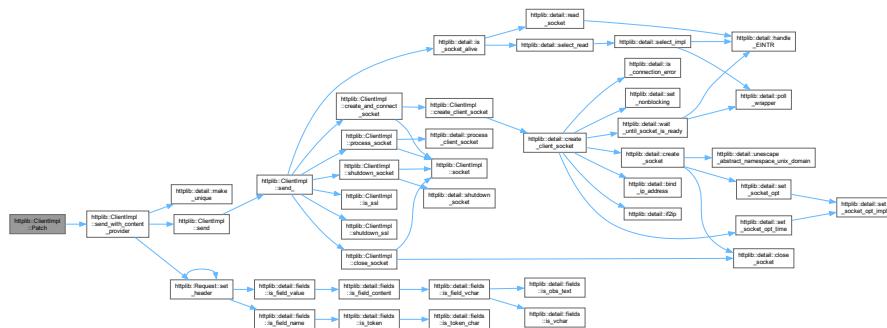
См. определение в файле `httpplib.h` строка 8748

```

08751 {
08752     return send_with_content_provider("PATCH", path, headers, body,
08753             content_length, nullptr, nullptr,
08754             content_type, progress);
08755 }

```

Граф вызовов:



6.5.3.46 Patch() [6/13]

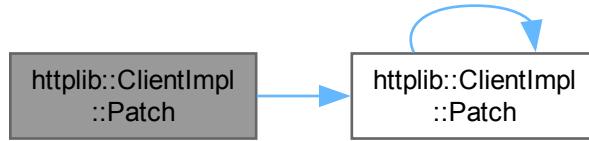
```
Result httplib::ClientImpl::Patch (
```

const std::string & path,
const Headers & headers,
const std::string & body,
const std::string & content_type) [inline]

См. определение в файле `httpplib.h` строка 8770

```
08772 }  
08773 return Patch(path, headers, body, content_type, nullptr);  
08774 }
```

Граф вызовов:



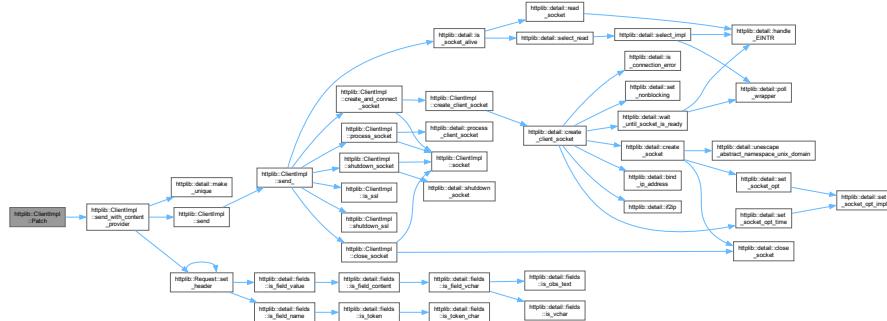
6.5.3.47 Patch() [7/13]

```
Result httplib::ClientImpl::Patch (const std::string & path, const Headers & headers, const std::string & body, const std::string & content_type, Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8776

```
08779             {  
08780     return send_with_content_provider("PATCH", path, headers, body.data(),  
08781                     body.size(), nullptr, nullptr, content_type,  
08782                     progress);  
08783 }
```

Граф вызовов:



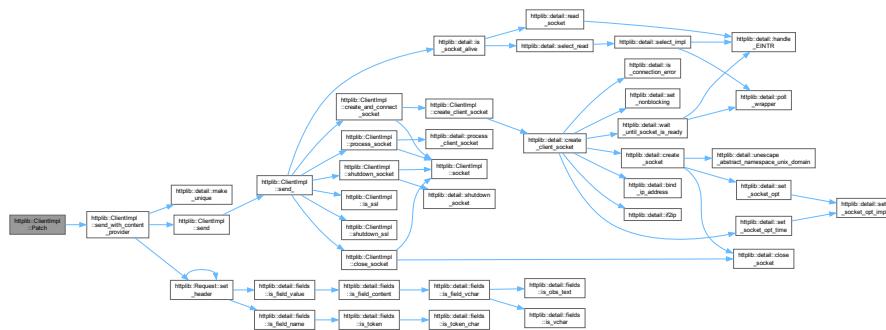
6.5.3.48 Patch() [8/13]

```
Result httpclient::ClientImpl::Patch (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpclient.h` строка 8807

```
08809     {
08810     return send_with_content_provider("PATCH", path, headers, nullptr, 0, nullptr,
08811             std::move(content_provider), content_type,
08812             nullptr);
08813 }
```

Граф вызовов:



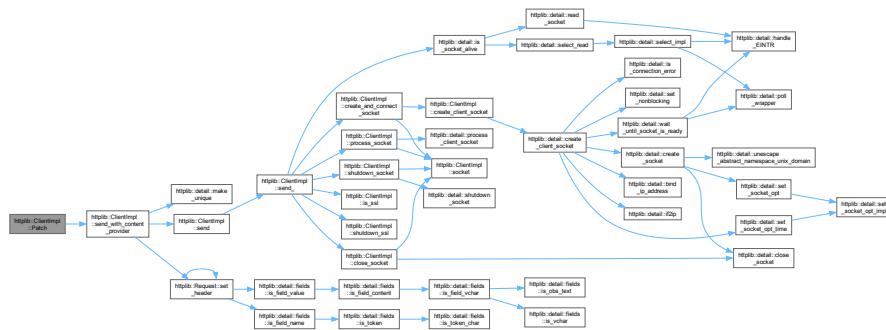
6.5.3.49 Patch() [9/13]

```
Result httpclient::ClientImpl::Patch (
    const std::string & path,
    const Headers & headers,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpclient.h` строка 8798

```
08801     {
08802     return send_with_content_provider("PATCH", path, headers, nullptr,
08803             content_length, std::move(content_provider),
08804             nullptr, content_type, nullptr);
08805 }
```

Граф вызовов:



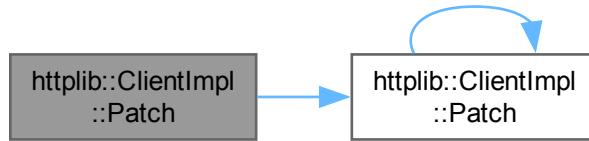
6.5.3.50 Patch() [10/13]

```
Result httplib::ClientImpl::Patch (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка [8757](#)

```
08759 {
08760     return Patch(path, Headers(), body, content_type);
08761 }
```

Граф вызовов:



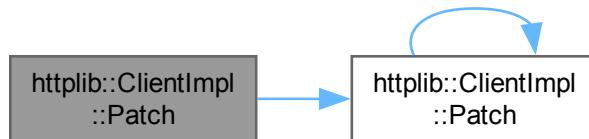
6.5.3.51 Patch() [11/13]

```
Result httplib::ClientImpl::Patch (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка [8763](#)

```
08766 {
08767     return Patch(path, Headers(), body, content_type, progress);
08768 }
```

Граф вызовов:



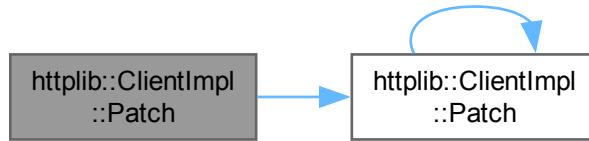
6.5.3.52 Patch() [12/13]

```
Result httplib::ClientImpl::Patch (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8792

```
08794     {
08795     return Patch(path, Headers(), std::move(content_provider), content_type);
08796 }
```

Граф вызовов:



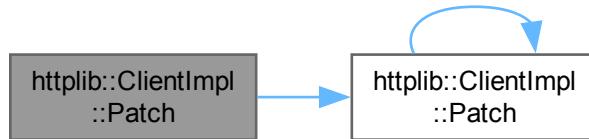
6.5.3.53 Patch() [13/13]

```
Result httplib::ClientImpl::Patch (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8785

```
08787     {
08788     return Patch(path, Headers(), content_length, std::move(content_provider),
08789             content_type);
08790 }
```

Граф вызовов:



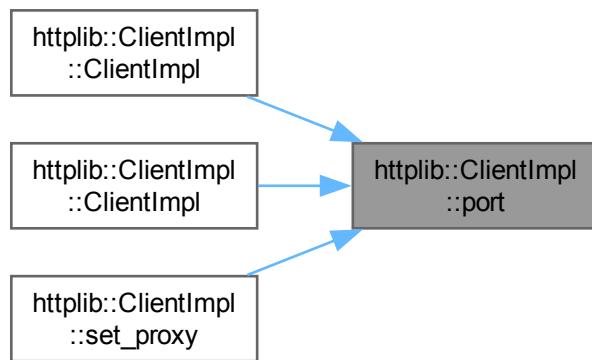
6.5.3.54 port()

```
int httpplib::ClientImpl::port () const [inline]
```

См. определение в файле [httpplib.h](#) строка 8936

```
08936 { return port_; }
```

Граф вызова функции:



6.5.3.55 Post() [1/20]

```
Result httpplib::ClientImpl::Post (
    const std::string & path) [inline]
```

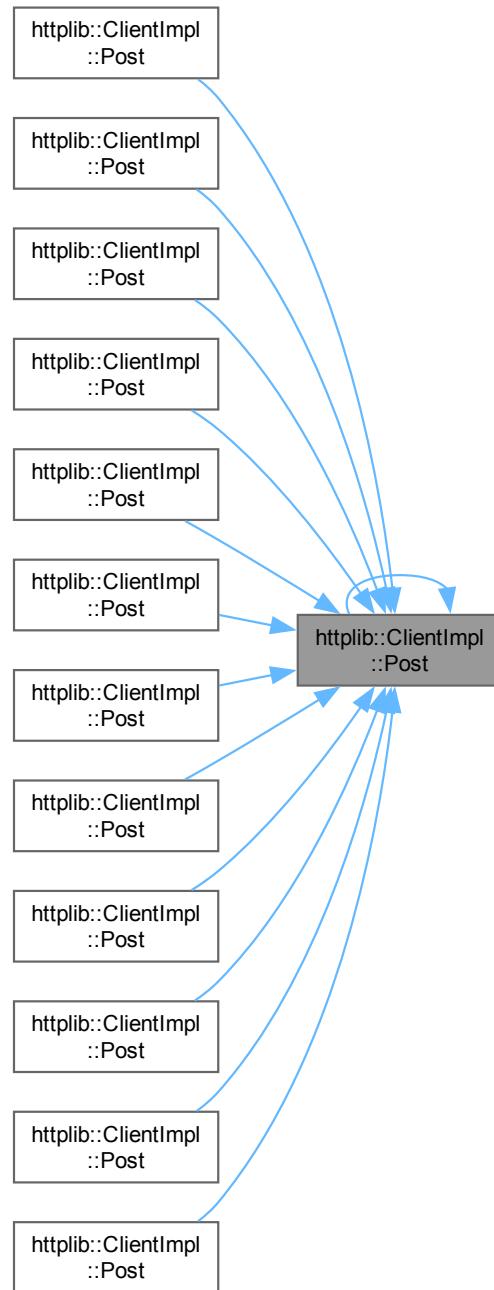
См. определение в файле [httpplib.h](#) строка 8441

```
08441 {
08442     return Post(path, std::string(), std::string());
08443 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.56 Post() [2/20]

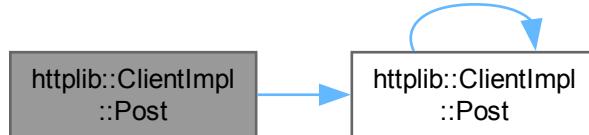
```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const char * body,
```

```
size_t content_length,
const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8450

```
08452     {
08453     return Post(path, Headers(), body, content_length, content_type, nullptr);
08454 }
```

Граф вызовов:



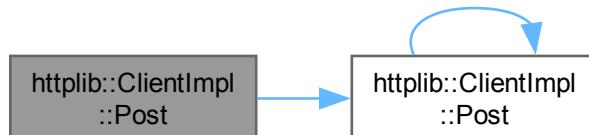
6.5.3.57 Post() [3/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers) [inline]
```

См. определение в файле [httplib.h](#) строка 8445

```
08446     {
08447     return Post(path, headers, nullptr, 0, std::string());
08448 }
```

Граф вызовов:



6.5.3.58 Post() [4/20]

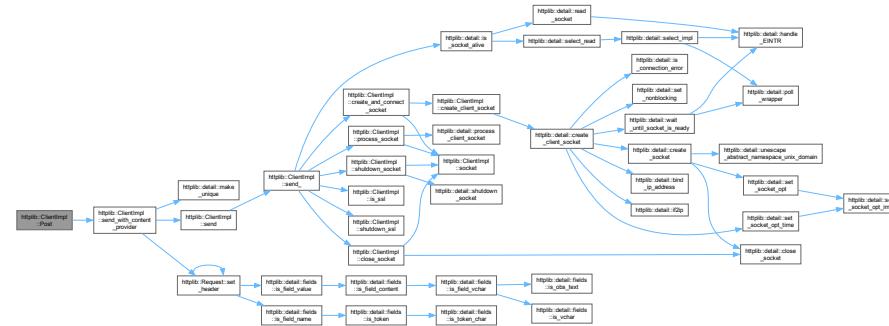
```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const char * body,
```

```
size_t content_length,  
const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8456

```
08458     {
08459     return send_with_content_provider("POST", path, headers, body, content_length,
08460                                         nullptr, nullptr, content_type, nullptr);
08461 }
```

Граф вызовов:



6.5.3.59 Post() [5/20]

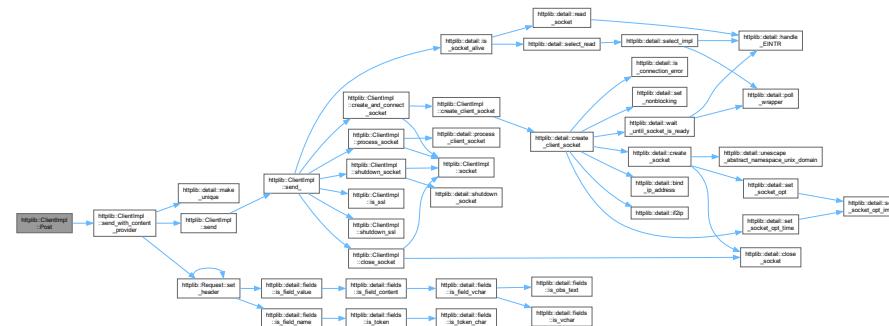
Result `httpplib::ClientImpl::Post`

```
const std::string & path,
const Headers & headers,
const char * body,
size_t content_length,
const std::string & content_type,
Progress progress) [inline]
```

См. определение в файле [httpplib.h](#) строка 8463

```
08466             {
08467     return send_with_content_provider("POST", path, headers, body, content_length,
08468                                         nullptr, nullptr, content_type, progress);
08469 }
```

Граф вызовов:



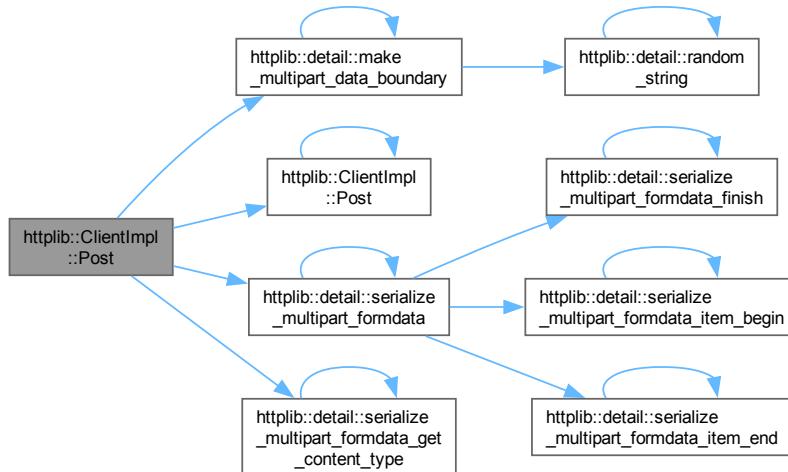
6.5.3.60 Post() [6/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле [httplib.h](#) строка 8551

```
08552     const auto &boundary = detail::make_multipart_data_boundary();
08553     const auto &content_type =
08554         detail::serialize_multipart_formdata_get_content_type(boundary);
08555     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08556     return Post(path, headers, body, content_type);
08557 }
08558 }
```

Граф вызовов:



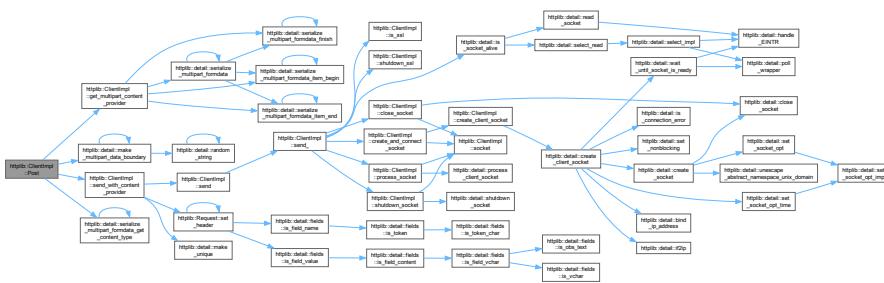
6.5.3.61 Post() [7/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const MultipartFormDataProviderItems & provider_items) [inline]
```

См. определение в файле [httplib.h](#) строка 8574

```
08576     const auto &boundary = detail::make_multipart_data_boundary();
08577     const auto &content_type =
08578         detail::serialize_multipart_formdata_get_content_type(boundary);
08579     return send_with_content_provider(
08580         "POST", path, headers, nullptr, 0, nullptr,
08581         get_multipart_content_provider(boundary, items, provider_items),
08582         content_type, nullptr);
08583 }
08584 }
```

Граф вызовов:



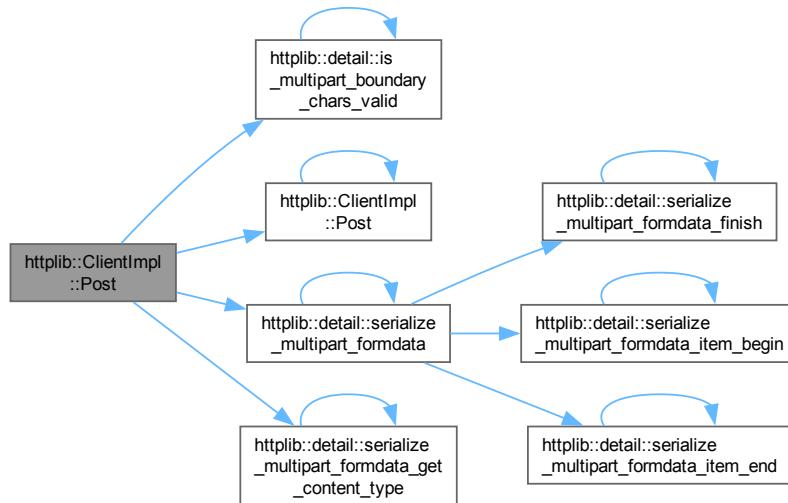
6.5.3.62 Post() [8/20]

```
Result http://ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const std::string & boundary) [inline]
```

См. определение в файле [http://ClientImpl.h](#) строка 8560

```
08562 {
08563     if (!detail::is_multipart_boundary_chars_valid(boundary)) {
08564         return Result{nullptr, Error::UnsupportedMultipartBoundaryChars};
08565     }
08566
08567     const auto &content_type =
08568         detail::serialize_multipart_formdata_get_content_type(boundary);
08569     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08570     return Post(path, headers, body, content_type);
08571 }
```

Граф вызовов:



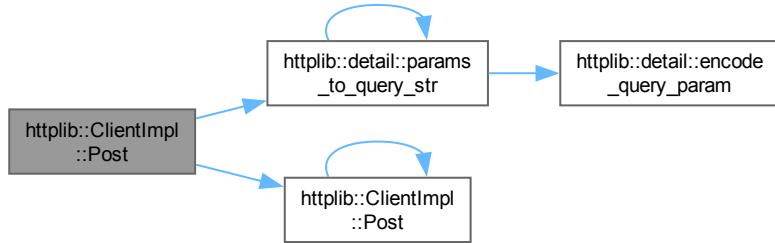
6.5.3.63 Post() [9/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const Params & params) [inline]
```

См. определение в файле [httplib.h](#) строка 8533

```
08534 {
08535     auto query = detail::params_to_query_str(params);
08536     return Post(path, headers, query, "application/x-www-form-urlencoded");
08537 }
```

Граф вызовов:



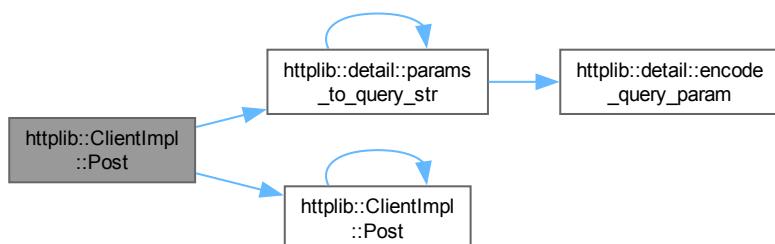
6.5.3.64 Post() [10/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const Params & params,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8539

```
08540 {
08541     auto query = detail::params_to_query_str(params);
08542     return Post(path, headers, query, "application/x-www-form-urlencoded",
08543             progress);
08544 }
```

Граф вызовов:



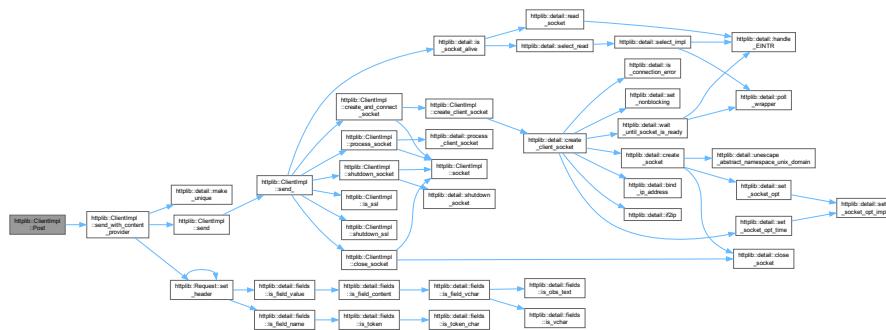
6.5.3.65 `Post()` [11/20]

```
Result httpclient::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле `httpclient.h` строка 8482

```
08484     {
08485     return send_with_content_provider("POST", path, headers, body.data(),
08486                                         body.size(), nullptr, nullptr, content_type,
08487                                         nullptr);
08488 }
```

Граф вызовов:

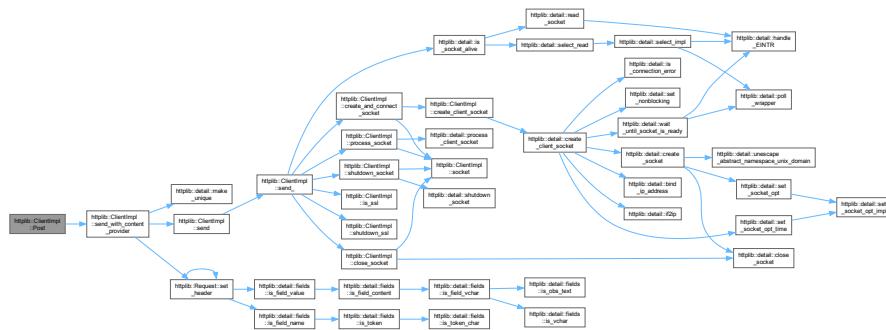
6.5.3.66 `Post()` [12/20]

```
Result httpclient::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле `httpclient.h` строка 8490

```
08493     {
08494     return send_with_content_provider("POST", path, headers, body.data(),
08495                                         body.size(), nullptr, nullptr, content_type,
08496                                         progress);
08497 }
```

Граф вызовов:



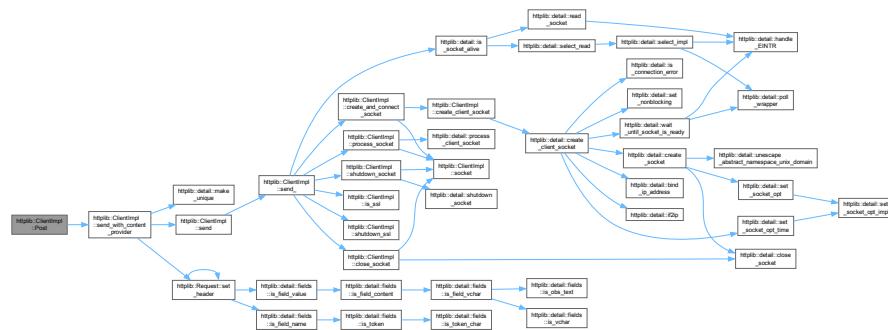
6.5.3.67 `Post()` [13/20]

```
Result httpclient::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpclient.h` строка 8525

```
08527     {
08528     return send_with_content_provider("POST", path, headers, nullptr, 0, nullptr,
08529                                     std::move(content_provider), content_type,
08530                                     nullptr);
08531 }
```

Граф вызовов:

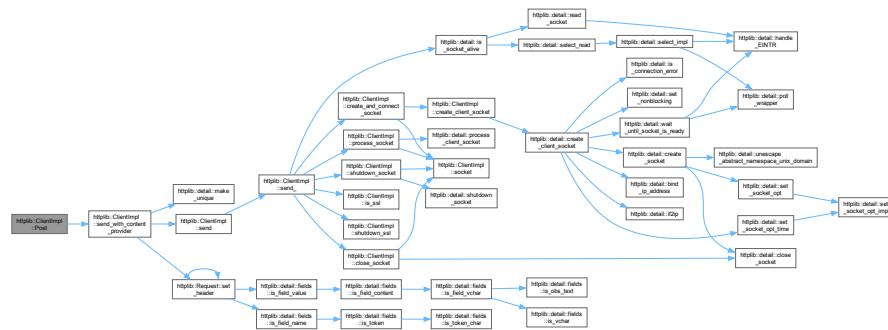
6.5.3.68 `Post()` [14/20]

```
Result httpclient::ClientImpl::Post (
    const std::string & path,
    const Headers & headers,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpclient.h` строка 8516

```
08519     {
08520     return send_with_content_provider("POST", path, headers, nullptr,
08521                                     content_length, std::move(content_provider),
08522                                     nullptr, content_type, nullptr);
08523 }
```

Граф вызовов:



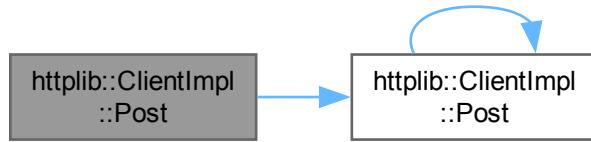
6.5.3.69 Post() [15/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле [httplib.h](#) строка 8546

```
08547 {  
08548     return Post(path, Headers(), items);  
08549 }
```

Граф вызовов:



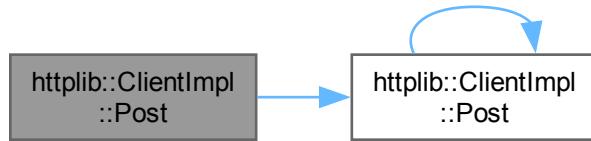
6.5.3.70 Post() [16/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const Params & params) [inline]
```

См. определение в файле [httplib.h](#) строка 8499

```
08499 {  
08500     return Post(path, Headers(), params);  
08501 }
```

Граф вызовов:



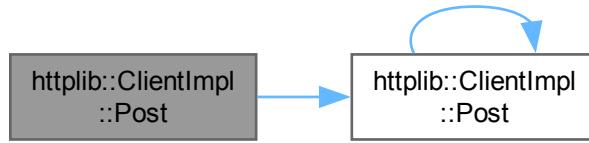
6.5.3.71 Post() [17/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8471

```
08472 {
08473     return Post(path, Headers(), body, content_type);
08474 }
```

Граф вызовов:



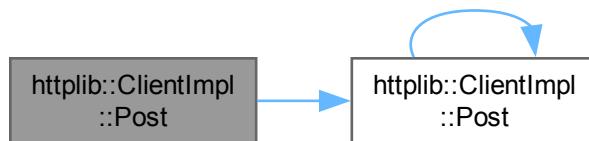
6.5.3.72 Post() [18/20]

```
Result httplib::ClientImpl::Post (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8476

```
08478 {
08479     return Post(path, Headers(), body, content_type, progress);
08480 }
```

Граф вызовов:



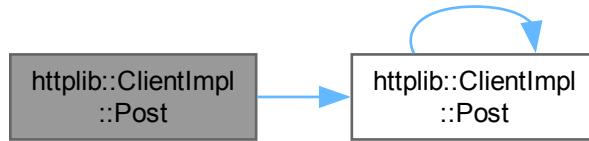
6.5.3.73 `Post()` [19/20]

```
Result httpplib::ClientImpl::Post (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 8510

```
08512 {
08513     return Post(path, Headers(), std::move(content_provider), content_type);
08514 }
```

Граф вызовов:

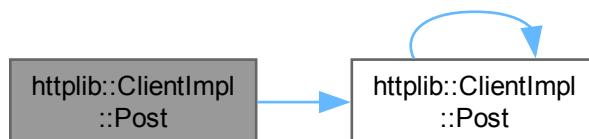
6.5.3.74 `Post()` [20/20]

```
Result httpplib::ClientImpl::Post (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 8503

```
08505 {
08506     return Post(path, Headers(), content_length, std::move(content_provider),
08507                 content_type);
08508 }
```

Граф вызовов:



6.5.3.75 `process_request()`

```
bool httpplib::ClientImpl::process_request (
    Stream & strm,
    Request & req,
    Response & res,
    bool close_connection,
    Error & error) [inline], [protected]
```

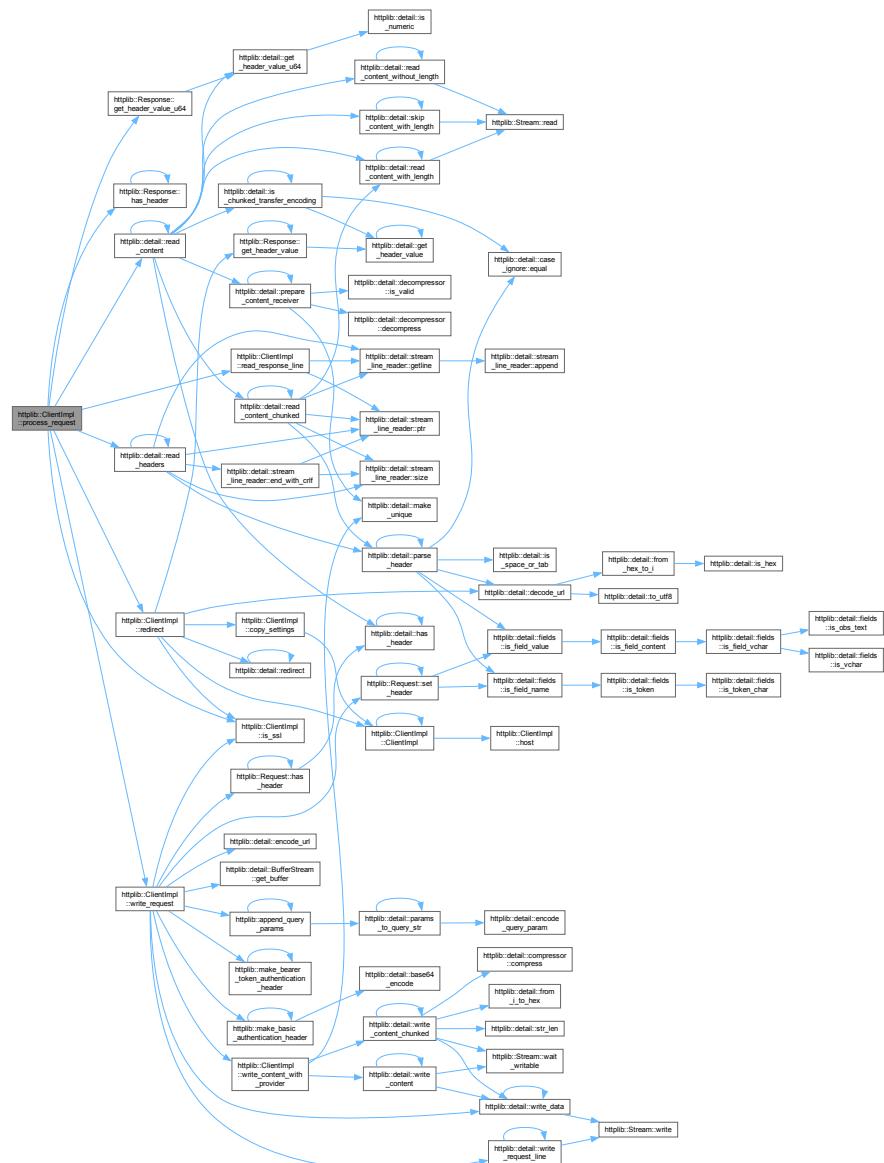
См. определение в файле `httpplib.h` строка 8153

```
08155 // Send request
08156 if (!write_request(strm, req, close_connection, error)) { return false; }
08158
08159 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
08160 if (is_ssl()) {
08161     auto is_proxy_enabled = !proxy_host_.empty() && proxy_port_ != -1;
08162     if (!is_proxy_enabled) {
08163         if (detail::is_ssl_peer_could_be_closed(socket_.ssl, socket_.sock)) {
08164             error = Error::SSLPeerCouldBeClosed_;
08165             return false;
08166         }
08167     }
08168 }
08169#endif
08170
08171 // Receive response and headers
08172 if (!read_response_line(strm, req, res) ||
08173     !detail::read_headers(strm, res.headers)) {
08174     error = Error::Read;
08175     return false;
08176 }
08177
08178 // Body
08179 if ((res.status != StatusCode::NoContent_204) && req.method != "HEAD" &&
08180     req.method != "CONNECT") {
08181     auto redirect = 300 < res.status && res.status < 400 &&
08182         res.status != StatusCode::NotModified_304 &&
08183         follow_location_;
08184
08185     if (req.response_handler && !redirect) {
08186         if (!req.response_handler(res)) {
08187             error = Error::Canceled;
08188             return false;
08189         }
08190     }
08191
08192     auto out =
08193         req.content_receiver
08194         ? static_cast<ContentReceiverWithProgress>(
08195             [&](const char *buf, size_t n, uint64_t off, uint64_t len) {
08196                 if (redirect) { return true; }
08197                 auto ret = req.content_receiver(buf, n, off, len);
08198                 if (!ret) { error = Error::Canceled; }
08199                 return ret;
08200             })
08201         : static_cast<ContentReceiverWithProgress>(
08202             [&](const char *buf, size_t n, uint64_t /*off*/,
08203                 uint64_t /*len*/) {
08204                 assert(res.body.size() + n <= res.body.max_size());
08205                 res.body.append(buf, n);
08206                 return true;
08207             });
08208
08209     auto progress = [&](uint64_t current, uint64_t total) {
08210         if (!req.progress || redirect) { return true; }
08211         auto ret = req.progress(current, total);
08212         if (!ret) { error = Error::Canceled; }
08213         return ret;
08214     };
08215
08216     if (res.has_header("Content-Length")) {
08217         if (!req.content_receiver) {
08218             auto len = res.get_header_value_u64("Content-Length");
08219             if (len > res.body.max_size()) {
08220                 error = Error::Read;
08221                 return false;
08222             }
08223             res.body.reserve(static_cast<size_t>(len));
08224         }
08225     }
08226 }
```

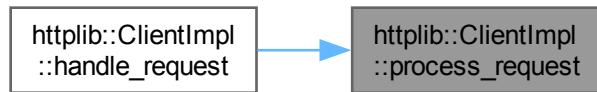
```

08227  if (res.status != StatusCode::NotModified_304) {
08228      int dummy_status;
08229      if (!detail::read_content(strm, res, (std::numeric_limits<size_t>::max)(),
08230          dummy_status, std::move(progress),
08231          std::move(out), decompress_)) {
08232          if (error != Error::Canceled) { error = Error::Read; }
08233          return false;
08234      }
08235  }
08236 }
08237
08238 // Log
08239 if (logger_) { logger_(req, res); }
08240
08241 return true;
08242 }
```

Граф вызовов:



Граф вызова функции:



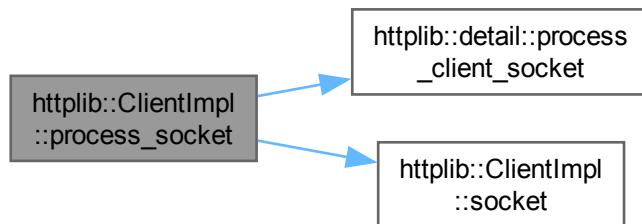
6.5.3.76 process_socket()

```
bool httplib::ClientImpl::process_socket (
    const Socket & socket,
    std::chrono::time_point<std::chrono::steady_clock> start_time,
    std::function< bool(Stream &strm)> callback) [inline], [private], [virtual]
```

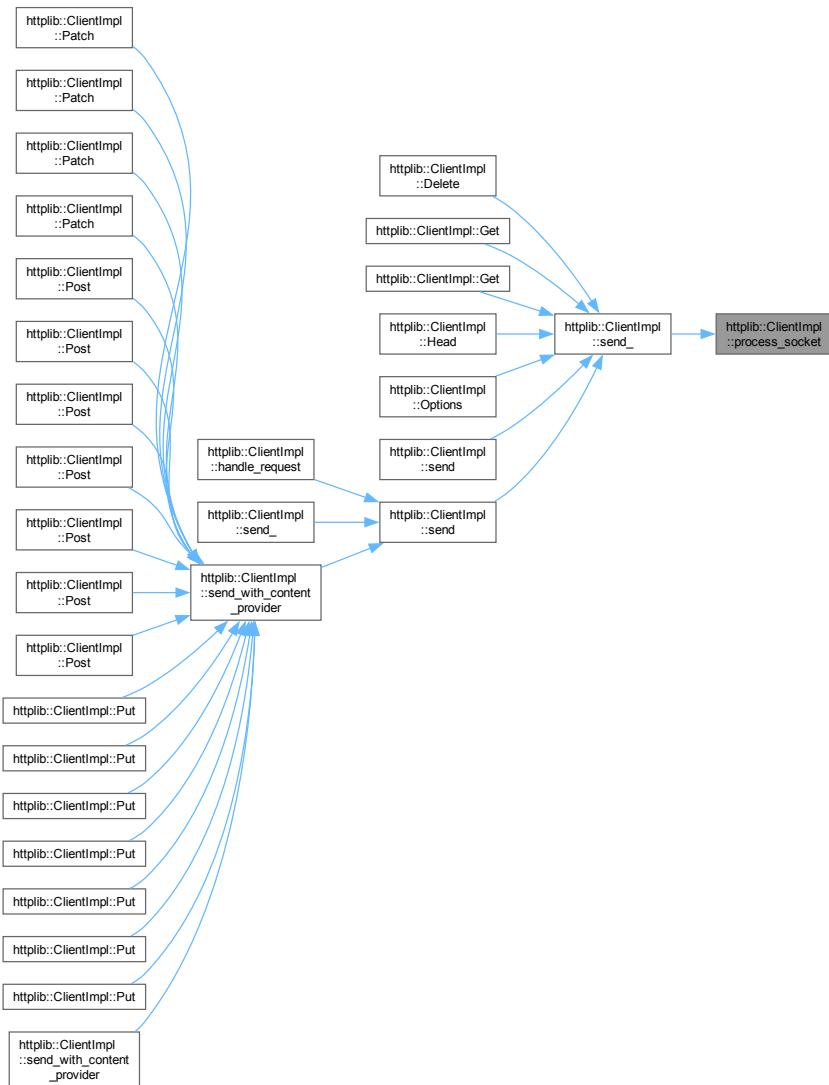
См. определение в файле [httplib.h](#) строка 8288

```
08291     {
08292     return detail::process_client_socket(
08293         socket.sock, read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
08294         write_timeout_usec_, max_timeout_msec_, start_time, std::move(callback));
08295 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.77 Put() [1/19]

Result `httpplib::ClientImpl::Put (`
 `const std::string & path)` [inline]

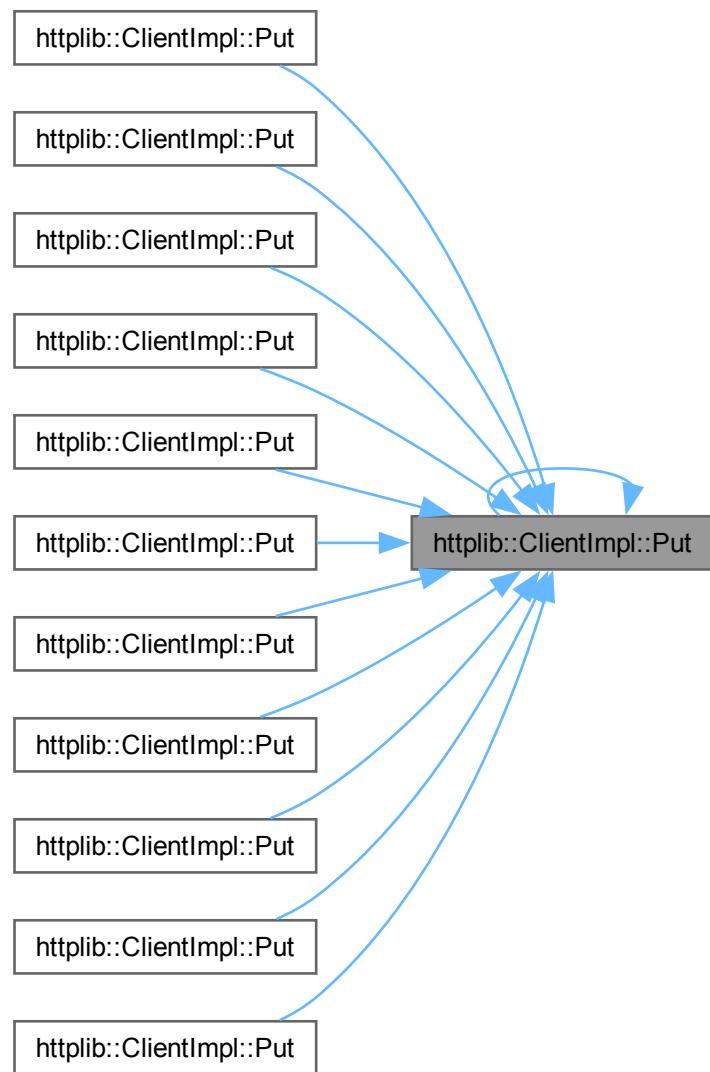
См. определение в файле [httplib.h](#) строка 8586

```
08586 {  
08587     return Put(path, std::string(), std::string());  
08588 }
```

Граф вызовов:



Граф вызова функции:



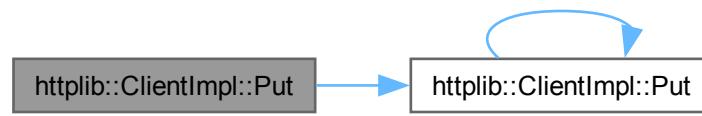
6.5.3.78 Put() [2/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8590

```
08592 {  
08593     return Put(path, Headers(), body, content_length, content_type);  
08594 }
```

Граф вызовов:



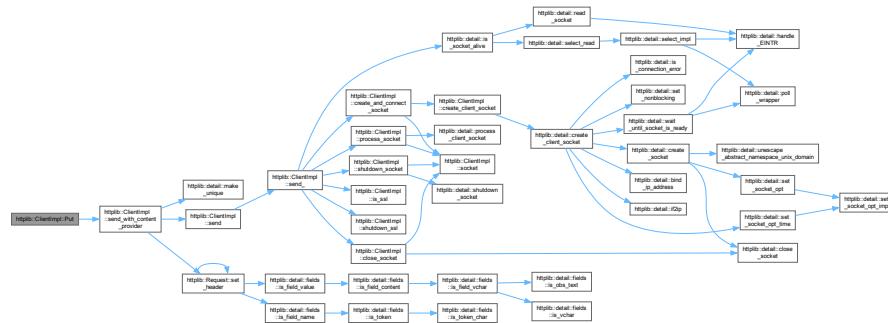
6.5.3.79 Put() [3/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const char * body,
    size_t content_length,
    const std::string & content_type) [inline]
```

См. определение в файле `httpplib.h` строка 8596

```
08598             {  
08599     return send_with_content_provider("PUT", path, headers, body, content_length,  
08600                     nullptr, nullptr, content_type, nullptr);  
08601 }
```

Граф вызовов:



6.5.3.80 Put() [4/19]

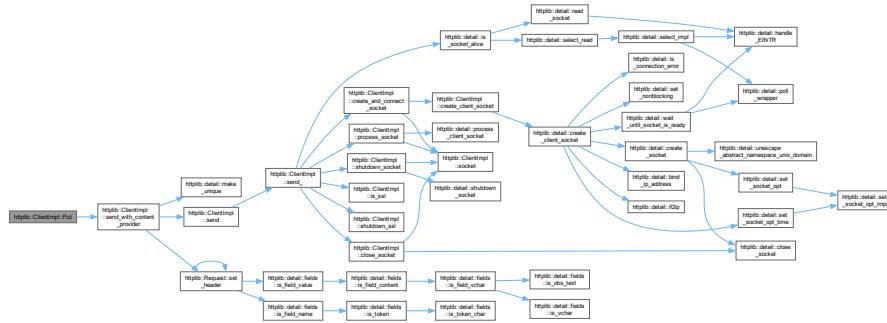
```
Result httplib::ClientImpl::Put (
```

const std::string & path,
const Headers & headers,
const char * body,
size_t content_length,
const std::string & content_type,
Progress progress) [inline]

См. определение в файле [httplib.h](#) строка 8603

```
08606             {
08607     return send_with_content_provider("PUT", path, headers, body, content_length,
08608                                         nullptr, nullptr, content_type, progress);
08609 }
```

Граф вызовов:



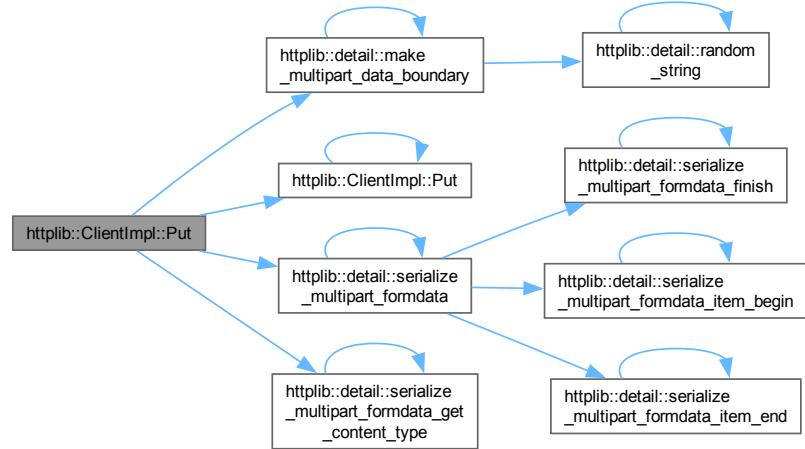
6.5.3.81 Put() [5/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле [httpplib.h](#) строка 8691

```
08692 {  
08693 const auto &boundary = detail::make_multipart_data_boundary();  
08694 const auto &content_type =  
08695     detail::serialize_multipartFormData_get_content_type(boundary);  
08696 const auto &body = detail::serialize_multipartFormData(items, boundary);  
08697 return Put(path, headers, body, content_type);  
08698 }
```

Граф вызовов:



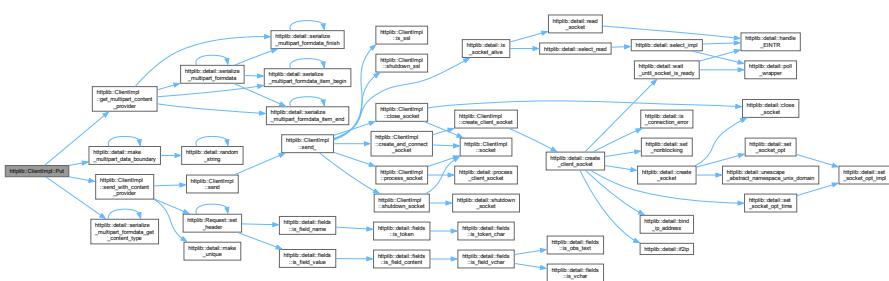
6.5.3.82 Put() [6/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const MultipartFormDataProviderItems & provider_items) [inline]
```

См. определение в файле `httplib.h` строка 8714

```
08716 {  
08717 const auto &boundary = detail::make_multipart_data_boundary();  
08718 const auto &content_type =  
08719     detail::serialize_multipart_formdata_get_content_type(boundary);  
08720 return send_with_content_provider(  
08721     "PUT", path, headers, nullptr, 0, nullptr,  
08722     get_multipart_content_provider(boundary, items, provider_items),  
08723     content_type, nullptr);  
08724 }
```

Граф вызовов:



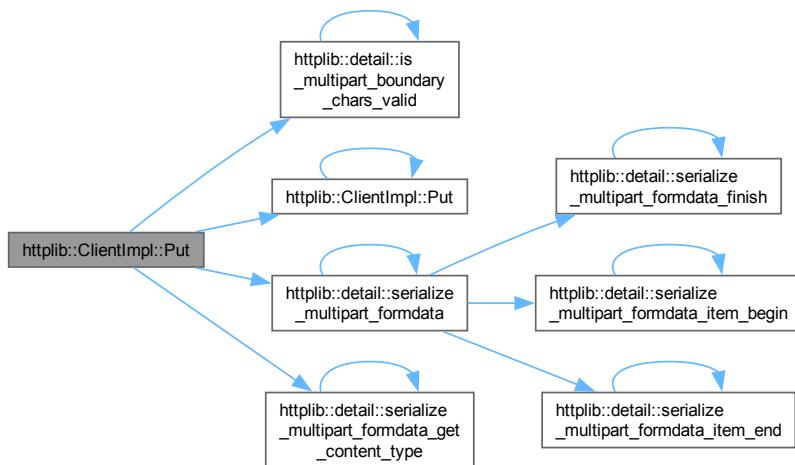
6.5.3.83 Put() [7/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const MultipartFormDataItems & items,
    const std::string & boundary) [inline]
```

См. определение в файле [httplib.h](#) строка 8700

```
08702     {
08703     if (!detail::is_multipart_boundary_chars_valid(boundary)) {
08704         return Result{nullptr, Error::UnsupportedMultipartBoundaryChars};
08705     }
08706
08707     const auto &content_type =
08708         detail::serialize_multipart_formdata_get_content_type(boundary);
08709     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08710     return Put(path, headers, body, content_type);
08711 }
```

Граф вызовов:



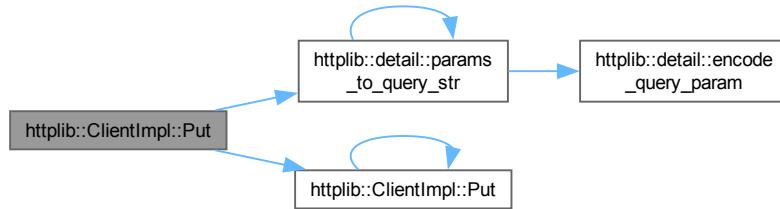
6.5.3.84 Put() [8/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const Params & params) [inline]
```

См. определение в файле [httplib.h](#) строка 8673

```
08674     {
08675     auto query = detail::params_to_query_str(params);
08676     return Put(path, headers, query, "application/x-www-form-urlencoded");
08677 }
```

Граф вызовов:



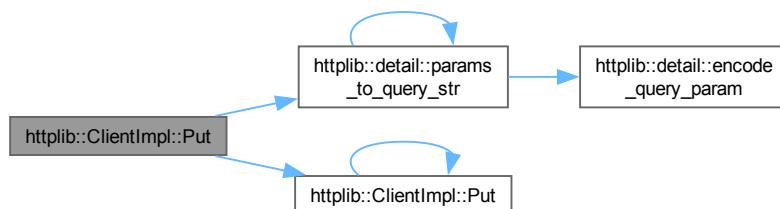
6.5.3.85 Put() [9/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const Params & params,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8679

```
08680
08681 auto query = detail::params_to_query_str(params);
08682 return Put(path, headers, query, "application/x-www-form-urlencoded",
08683 progress);
08684 }
```

Граф вызовов:



6.5.3.86 Put() [10/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type) [inline]
```

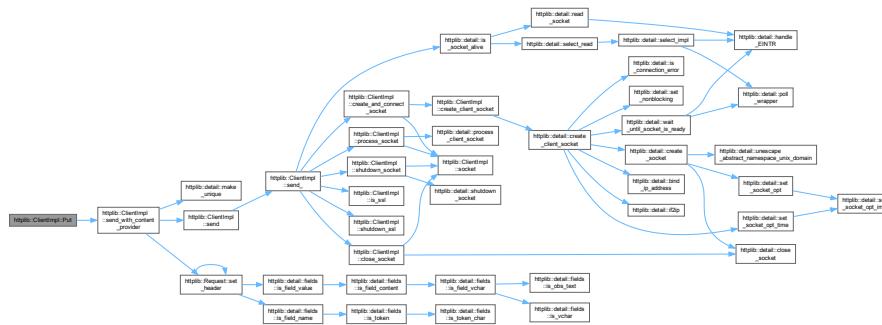
См. определение в файле [httplib.h](#) строка 8622

```

08624
08625  return send_with_content_provider("PUT", path, headers, body.data(),
08626                      body.size(), nullptr, nullptr, content_type,
08627                      nullptr);
08628 }

```

Граф вызовов:



6.5.3.87 Put() [11/19]

```

Result http://ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]

```

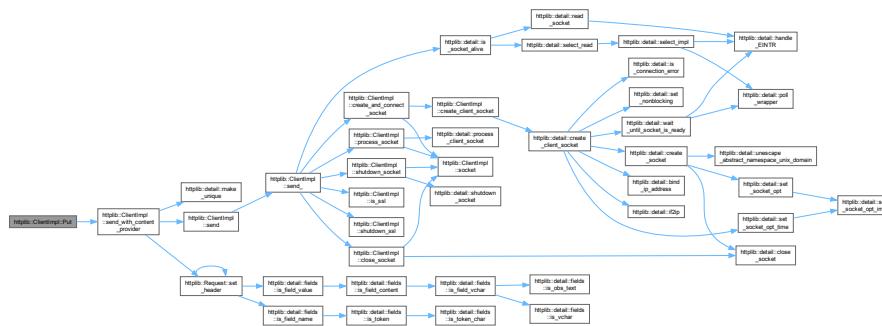
См. определение в файле `http://ClientImpl.h` строка 8630

```

08633
08634  return send_with_content_provider("PUT", path, headers, body.data(),
08635                      body.size(), nullptr, nullptr, content_type,
08636                      progress);
08637 }

```

Граф вызовов:



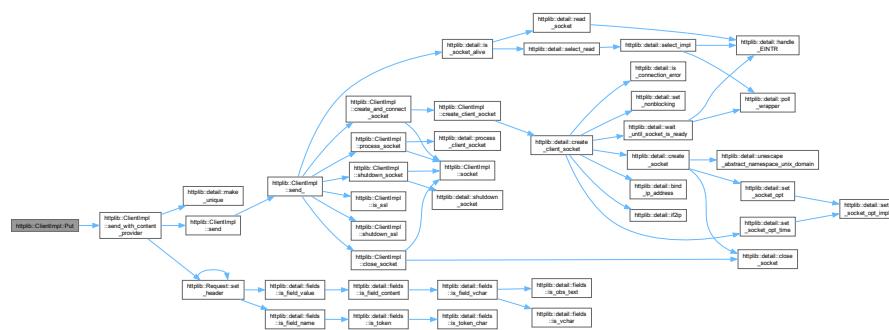
6.5.3.88 Put() [12/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Headers & headers,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8661

```
08663 {  
08664     return send_with_content_provider("PUT", path, headers, nullptr, 0, nullptr,  
08665                     std::move(content_provider), content_type,  
08666                     nullptr);  
08667 }
```

Граф вызовов:



6.5.3.89 Put() [13/19]

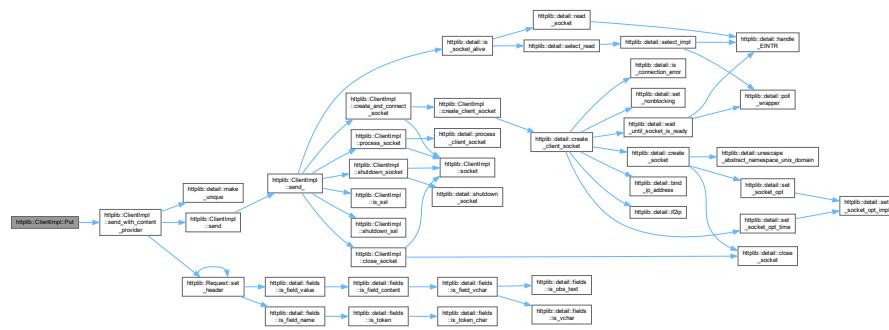
```
Result httplib::ClientImpl::Put (
```

const std::string & path,
const Headers & headers,
size_t content_length,
ContentProvider content_provider,
const std::string & content_type) [inline]

См. определение в файле `httpplib.h` строка 8652

```
08655             {  
08656     return send_with_content_provider("PUT", path, headers, nullptr,  
08657                     content_length, std::move(content_provider),  
08658                     nullptr, content_type, nullptr);  
08659 }
```

Граф вызовов:



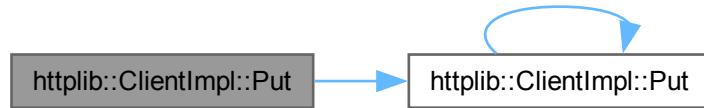
6.5.3.90 Put() [14/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const MultipartFormDataItems & items) [inline]
```

См. определение в файле [httplib.h](#) строка 8686

```
08687 {  
08688     return Put(path, Headers(), items);  
08689 }
```

Граф вызовов:



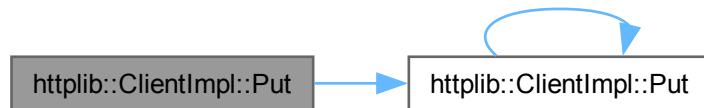
6.5.3.91 Put() [15/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const Params & params) [inline]
```

См. определение в файле [httplib.h](#) строка 8669

```
08669 {  
08670     return Put(path, Headers(), params);  
08671 }
```

Граф вызовов:



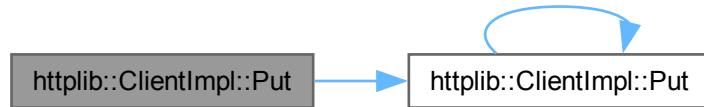
6.5.3.92 Put() [16/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const std::string & body,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8611

```
08612 {
08613     return Put(path, Headers(), body, content_type);
08614 }
```

Граф вызовов:



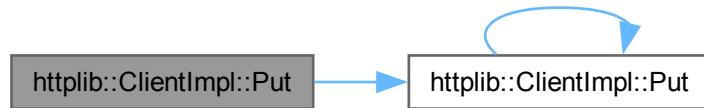
6.5.3.93 Put() [17/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    const std::string & body,
    const std::string & content_type,
    Progress progress) [inline]
```

См. определение в файле [httplib.h](#) строка 8616

```
08618 {
08619     return Put(path, Headers(), body, content_type, progress);
08620 }
```

Граф вызовов:



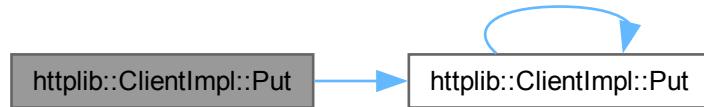
6.5.3.94 Put() [18/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    ContentProviderWithoutLength content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8646

```
08648 {
08649     return Put(path, Headers(), std::move(content_provider), content_type);
08650 }
```

Граф вызовов:



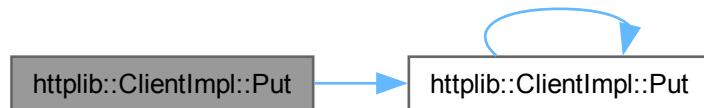
6.5.3.95 Put() [19/19]

```
Result httplib::ClientImpl::Put (
    const std::string & path,
    size_t content_length,
    ContentProvider content_provider,
    const std::string & content_type) [inline]
```

См. определение в файле [httplib.h](#) строка 8639

```
08641 {
08642     return Put(path, Headers(), content_length, std::move(content_provider),
08643             content_type);
08644 }
```

Граф вызовов:



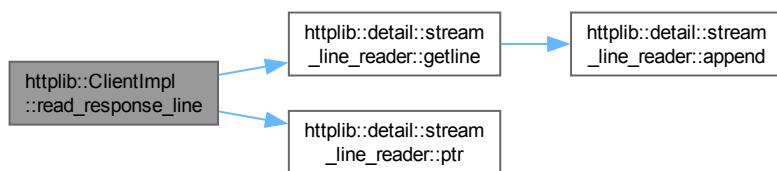
6.5.3.96 `read_response_line()`

```
bool httpplib::ClientImpl::read_response_line (
    Stream & strm,
    const Request & req,
    Response & res) const [inline], [private]
```

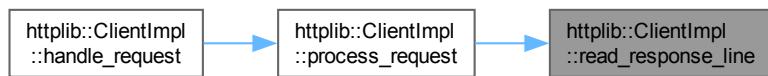
См. определение в файле `httpplib.h` строка 7587

```
07588 {
07589     std::array<char, 2048> buf{};
07590
07591     detail::stream_line_reader line_reader(strm, buf.data(), buf.size());
07592
07593     if (!line_reader.getline()) { return false; }
07594
07595 #ifndef CPPHTTPPLIB_ALLOW_LF_AS_LINE_TERMINATOR
07596     thread_local const std::regex re("(HTTP/1\\.\\.[01]) (\\d{3})(?: (.*)?)?\\r?\\n");
07597 #else
07598     thread_local const std::regex re("(HTTP/1\\.\\.[01]) (\\d{3})(?: (.*)?)?\\r\\n");
07599 #endif
07600
07601     std::cmatch m;
07602     if (!std::regex_match(line_reader.ptr(), m, re)) {
07603         return req.method == "CONNECT";
07604     }
07605     res.version = std::string(m[1]);
07606     res.status = std::stoi(std::string(m[2]));
07607     res.reason = std::string(m[3]);
07608
07609 // Ignore '100 Continue'
07610     while (res.status == StatusCode::Continue_100) {
07611         if (!line_reader.getline()) { return false; } // CRLF
07612         if (!line_reader.getline()) { return false; } // next response line
07613
07614         if (!std::regex_match(line_reader.ptr(), m, re)) { return false; }
07615         res.version = std::string(m[1]);
07616         res.status = std::stoi(std::string(m[2]));
07617         res.reason = std::string(m[3]);
07618     }
07619
07620     return true;
07621 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.97 redirect()

```
bool httplib::ClientImpl::redirect (
    Request & req,
    Response & res,
    Error & error) [inline], [private]
```

См. определение в файле [httplib.h](#) строка 7821

```
07821 {  

07822     if (req.redirect_count == 0) {  

07823         error = Error::ExceedRedirectCount;  

07824         return false;  

07825     }  

07826  

07827     auto location = res.get_header_value("location");  

07828     if (location.empty()) { return false; }  

07829  

07830     thread_local const std::regex re(  

07831         R"((?:https?://)?(?:\|([a-zA-F\d]+)\|)([^:/#]+)(?:(\d+))?)?([^\?#]*)(\?[^\#]*)(?:\#.*))?" );  

07832  

07833     std::smatch m;  

07834     if (!std::regex_match(location, m, re)) { return false; }  

07835  

07836     auto scheme = is_ssl() ? "https" : "http";  

07837  

07838     auto next_scheme = m[1].str();  

07839     auto next_host = m[2].str();  

07840     if (next_host.empty()) { next_host = m[3].str(); }  

07841     auto port_str = m[4].str();  

07842     auto next_path = m[5].str();  

07843     auto next_query = m[6].str();  

07844  

07845     auto next_port = port;  

07846     if (!port_str.empty()) {  

07847         next_port = std::stoi(port_str);  

07848     } else if (!next_scheme.empty()) {  

07849         next_port = next_scheme == "https" ? 443 : 80;  

07850     }  

07851  

07852     if (next_scheme.empty()) { next_scheme = scheme; }  

07853     if (next_host.empty()) { next_host = host; }  

07854     if (next_path.empty()) { next_path = "/"; }  

07855  

07856     auto path = detail::decode_url(next_path, true) + next_query;  

07857  

07858     if (next_scheme == scheme && next_host == host_ && next_port == port_) {  

07859         return detail::redirect(*this, req, res, path, location, error);  

07860     } else {  

07861         if (next_scheme == "https") {  

07862             #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT  

07863                 SSLClient cli(next_host, next_port);  

07864                 cli.copy_settings(*this);  

07865                 if (ca_cert_store_) { cli.set_ca_cert_store(ca_cert_store_); }  

07866                 return detail::redirect(cli, req, res, path, location, error);  

07867         } else {  

07868             return false;  

07869         }  

07870     }  

07871     ClientImpl cli(next_host, next_port);  

07872     cli.copy_settings(*this);  

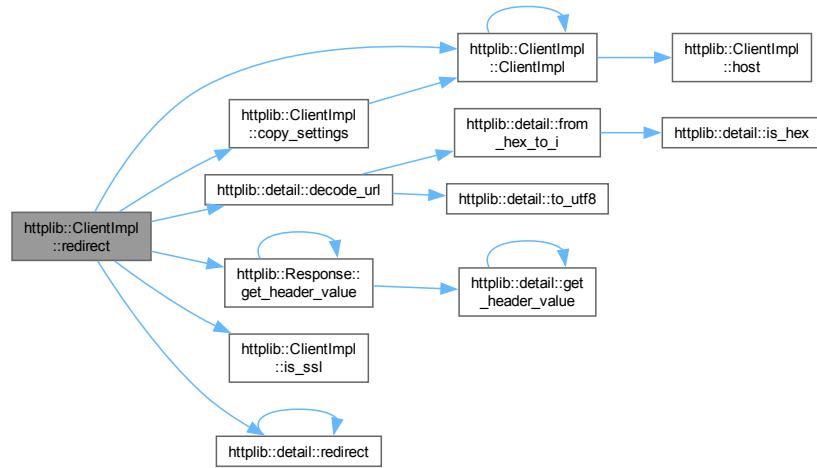
07873     return detail::redirect(cli, req, res, path, location, error);  

07874 }  

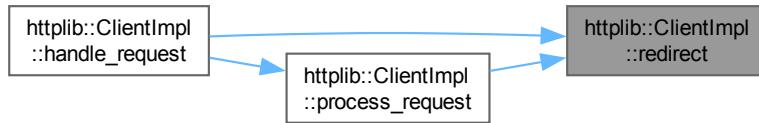
07875 }  

07876 }
```

Граф вызовов:



Граф вызова функции:



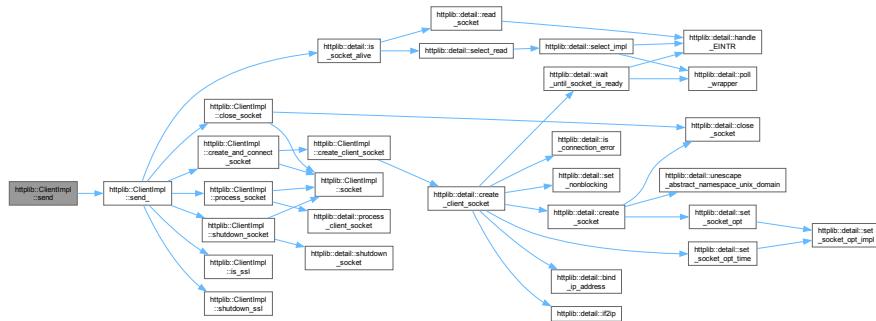
6.5.3.98 send() [1/2]

```
Result httplib::ClientImpl::send (
    const Request & req) [inline]
```

См. определение в файле [httplib.h](#) строка [7732](#)

```
07732 {
07733     auto req2 = req;
07734     return send_(std::move(req2));
07735 }
```

Граф вызовов:



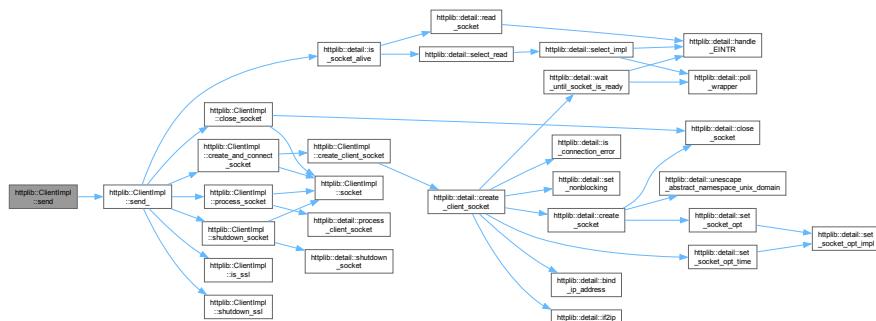
6.5.3.99 send() [2/2]

```
bool httplib::ClientImpl::send (
    Request & req,
    Response & res,
    Error & error) [inline]
```

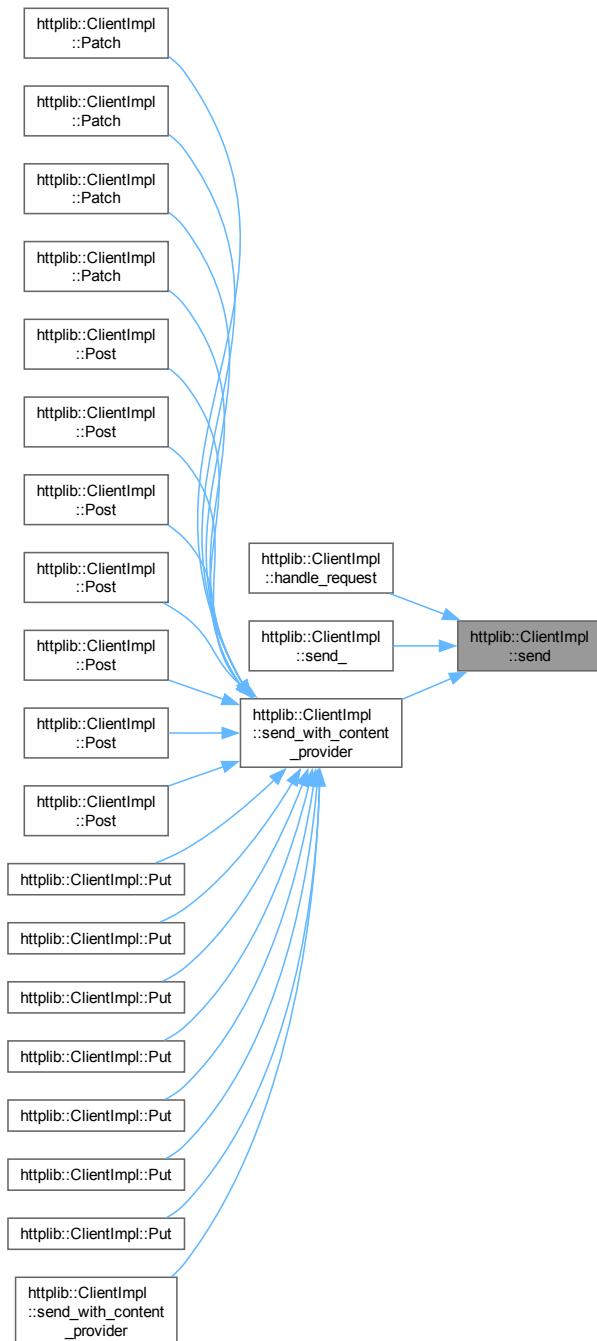
См. определение в файле [httpplib.h](#) строка 7623

```
07623     {  
07624     std::lock_guard<std::recursive_mutex> request_mutex_guard(request_mutex_);  
07625     auto ret = send_(req, res, error);  
07626     if (error == Error::SSLPeerCouldBeClosed_) {  
07627         assert(!ret);  
07628         ret = send_(req, res, error);  
07629     }  
07630     return ret;  
07631 }
```

Граф вызовов:



Граф вызова функции:



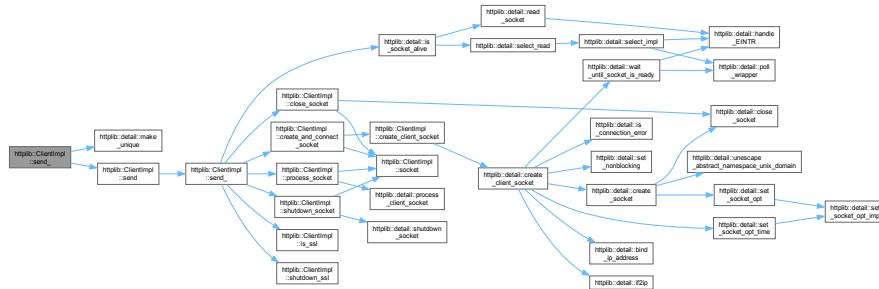
6.5.3.100 send_() [1/2]

Result `httpplib::ClientImpl::send_ (`
Request && `req)` [inline], [private]

См. определение в файле `httpplib.h` строка 7737

```
07737     {
07738     auto res = detail::make_unique<Response>();
07739     auto error = Error::Success;
07740     auto ret = send(req, *res, error);
07741     return Result{ret ? std::move(res) : nullptr, error, std::move(req.headers)};
07742 }
```

Граф вызовов:



6.5.3.101 send_() [2/2]

```
bool httplib::ClientImpl::send_(
    Request & req,
    Response & res,
    Error & error) [inline], [private]
```

См. определение в файле `httpplib.h` строка 7633

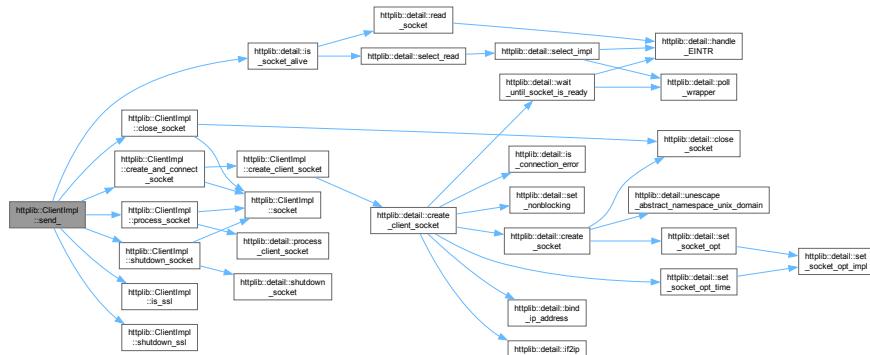
```
07633 {  
07634     std::lock_guard<std::mutex> guard(socket_mutex_);  
07635  
07636 // Set this to false immediately - if it ever gets set to true by the end of  
07637 // the request, we know another thread instructed us to close the socket.  
07638 socket_should_be_closed_when_request_is_done_ = false;  
07639  
07640     auto is_alive = false;  
07641     if (socket_.is_open()) {  
07642         is_alive = detail::is_socket_alive(socket_.sock);  
07643     }  
07644  
07645 #ifndef CPPHTTPPLIB_OPENSSL_SUPPORT  
07646     if (is_alive && is_ssl()) {  
07647         if (!detail::is_ssl_peer_could_be_closed(socket_.ssl, socket_.sock)) {  
07648             is_alive = false;  
07649         }  
07650     }  
07651 #endif  
07652  
07653     if (!is_alive) {  
07654         // Attempt to avoid sigpipe by shutting down non-gracefully if it seems  
07655         // like the other side has already closed the connection. Also, there  
07656         // cannot be any requests in flight from other threads since we locked  
07657         // request_mutex_, so safe to close everything immediately  
07658         const bool shutdown_gracefully = false;  
07659         shutdown_ssl(socket_, shutdown_gracefully);  
07660         shutdown_socket(socket_);  
07661         close_socket(socket_);  
07662     }  
07663 }  
07664  
07665     if (!is_alive) {  
07666         if (!create_and_connect_socket(socket_, error)) { return false; }  
07667  
07668 #ifndef CPPHTTPPLIB_OPENSSL_SUPPORT  
07669     // TODO: refactoring  
07670     if (is_ssl()) {  
07671         auto &scli = static_cast<SSLClient &>(*this);  
07672         if (!proxy_host_.empty() && proxy_port_ != -1) {  
07673             auto success = false;
```

```

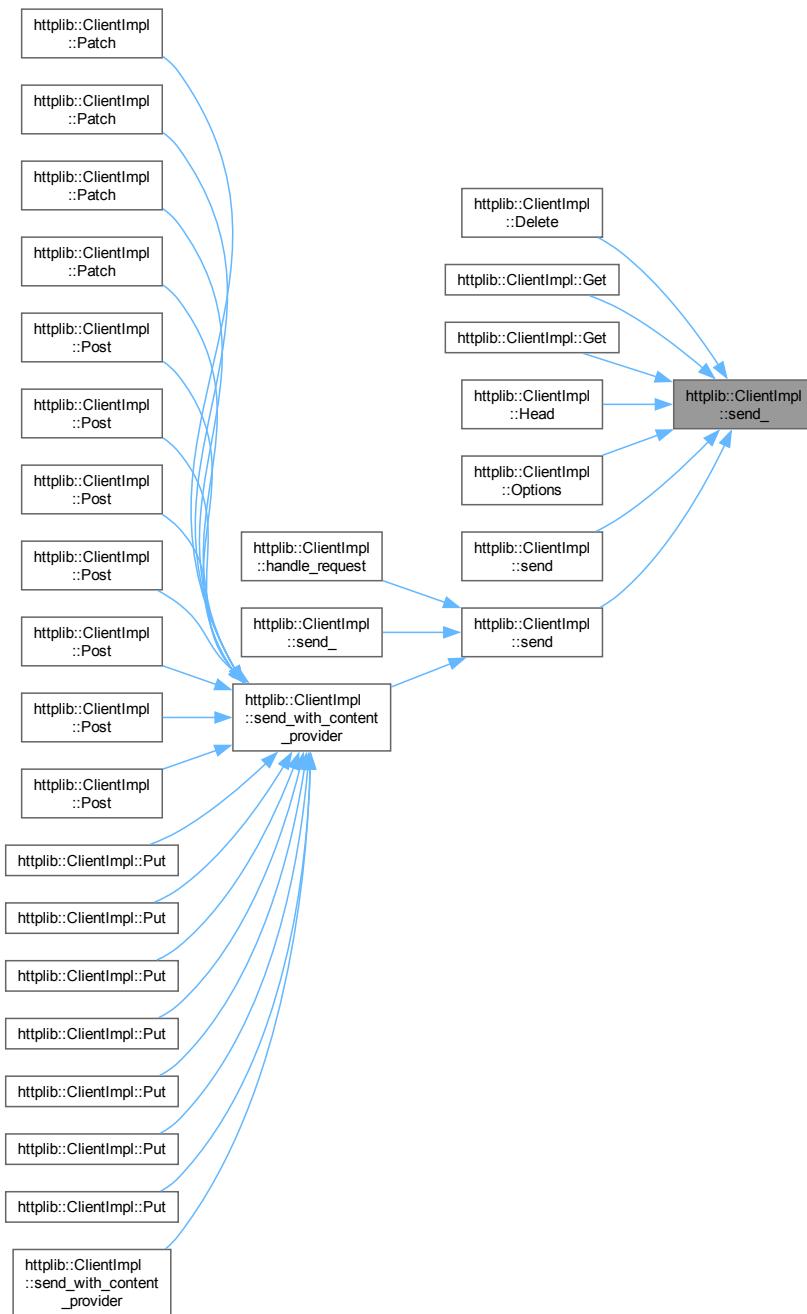
07674     if (!scli.connect_with_proxy(socket_, req.start_time_, res, success,
07675             error)) {
07676         return success;
07677     }
07678 }
07679
07680     if (!scli.initialize_ssl(socket_, error)) { return false; }
07681 }
07682 #endif
07683 }
07684
07685 // Mark the current socket as being in use so that it cannot be closed by
07686 // anyone else while this request is ongoing, even though we will be
07687 // releasing the mutex.
07688 if (socket_requests_in_flight_ > 1) {
07689     assert(socket_requests_are_from_thread_ == std::this_thread::get_id());
07690 }
07691 socket_requests_in_flight_ += 1;
07692 socket_requests_are_from_thread_ = std::this_thread::get_id();
07693 }
07694
07695 for (const auto &header : default_headers_) {
07696     if (req.headers.find(header.first) == req.headers.end()) {
07697         req.headers.insert(header);
07698     }
07699 }
07700
07701 auto ret = false;
07702 auto close_connection = !keep_alive_;
07703
07704 auto se = detail::scope_exit([&]{
07705     // Briefly lock mutex in order to mark that a request is no longer ongoing
07706     std::lock_guard<std::mutex> guard(socket_mutex_);
07707     socket_requests_in_flight_ -= 1;
07708     if (socket_requests_in_flight_ <= 0) {
07709         assert(socket_requests_in_flight_ == 0);
07710         socket_requests_are_from_thread_ = std::thread::id();
07711     }
07712
07713     if (socket_should_be_closed_when_request_is_done_ || close_connection ||
07714         !ret) {
07715         shutdown_ssl(socket_, true);
07716         shutdown_socket(socket_);
07717         close_socket(socket_);
07718     }
07719 });
07720
07721 ret = process_socket(socket_, req.start_time_, [&](Stream &strm) {
07722     return handle_request(strm, req, res, close_connection, error);
07723 });
07724
07725 if (!ret) {
07726     if (error == Error::Success) { error = Error::Unknown; }
07727 }
07728
07729 return ret;
07730 }

```

Граф вызовов:



Граф вызова функции:



6.5.3.102 send_with_content_provider() [1/2]

```
Result httplib::ClientImpl::send_with_content_provider (
```

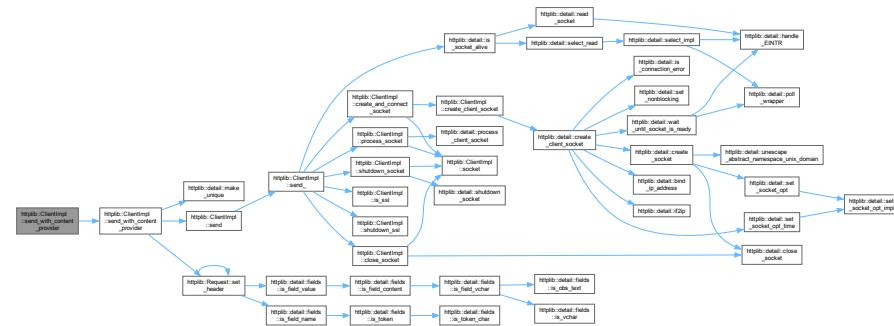
const std::string & method,	
const std::string & path,	
const Headers & headers,	
const char * body,	

```
size_t content_length,  
ContentProvider content_provider,  
ContentProviderWithoutLength content_provider_without_length,  
const std::string & content_type,  
Progress progress) [inline], [private]
```

См. определение в файле `httpplib.h` строка 8124

```
08128 {  
08129 Request req;  
08130 req.method = method;  
08131 req.headers = headers;  
08132 req.path = path;  
08133 req.progress = progress;  
08134 if (max_timeout_msec > 0) {  
08135     req.start_time_ = std::chrono::steady_clock::now();  
08136 }  
08137  
08138 auto error = Error::Success;  
08139  
08140 auto res = send_with_content_provider(  
08141     req, body, content_length, std::move(content_provider),  
08142     std::move(content_provider_without_length), content_type, error);  
08143  
08144 return Result{std::move(res), error, std::move(req.headers)};  
08145 }
```

Граф вызовов:



6.5.3.103 send_with_content_provider() [2/2]

```
std::unique_ptr<Response> httplib::ClientImpl::send_with_content_provider (
    Request &req,
    const char * body,
    size_t content_length,
    ContentProvider content_provider,
    ContentProviderWithoutLength content_provider_without_length,
    const std::string & content_type,
    Error &error) [inline], [private]
```

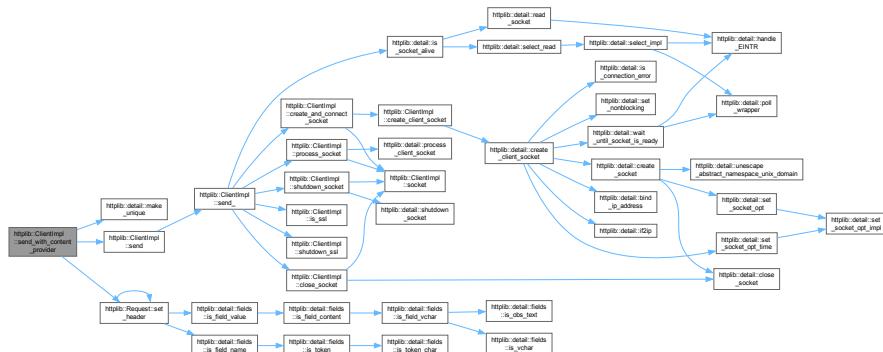
См. определение в файле `httpplib.h` строка 8045

```
08049         {
08050     if (!content_type.empty()) { req.set_header("Content-Type", content_type); }
08051
08052 #ifdef CPPHTTPPLIB_ZLIB_SUPPORT
08053     if (compress_) { req.set_header("Content-Encoding", "gzip"); }
08054 #endif
08055
08056 #ifdef CPPHTTPPLIB_ZLIB_SUPPORT
08057     if (compress_ && !content_provider_without_length) {
08058         // TODO: Brotli support
08059         detail::gzip_compressor compressor;
08060     }
```

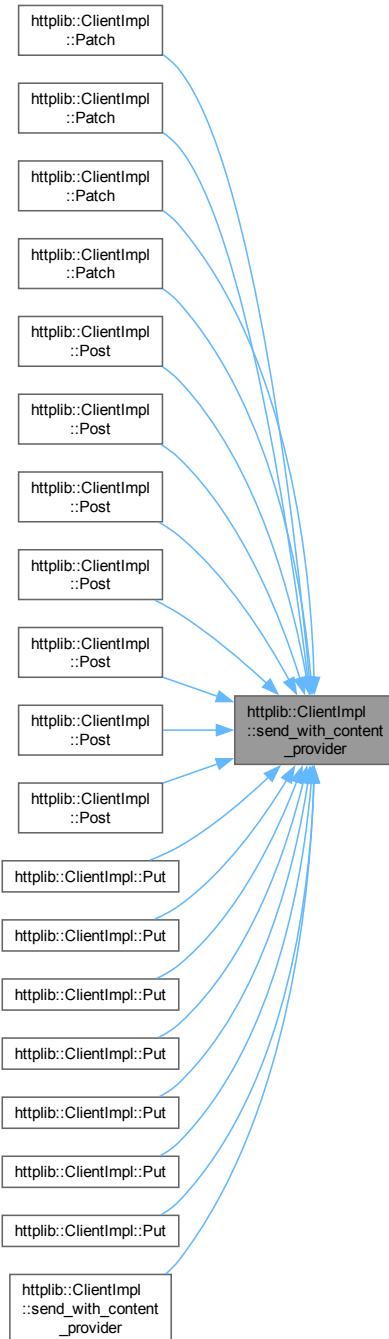
```

08061 if (content_provider) {
08062     auto ok = true;
08063     size_t offset = 0;
08064     DataSink data_sink;
08065
08066     data_sink.write = [&](const char *data, size_t data_len) -> bool {
08067         if (!ok) {
08068             auto last = offset + data_len == content_length;
08069
08070             auto ret = compressor.compress(
08071                 data, data_len, last,
08072                 [&](const char *compressed_data, size_t compressed_data_len) {
08073                     req.body.append(compressed_data, compressed_data_len);
08074                     return true;
08075                 });
08076
08077             if (ret) {
08078                 offset += data_len;
08079             } else {
08080                 ok = false;
08081             }
08082         }
08083         return ok;
08084     };
08085
08086     while (ok && offset < content_length) {
08087         if (!content_provider(offset, content_length - offset, data_sink)) {
08088             error = Error::Canceled;
08089             return nullptr;
08090         }
08091     }
08092 } else {
08093     if (!compressor.compress(body, content_length, true,
08094                             [&](const char *data, size_t data_len) {
08095                                 req.body.append(data, data_len);
08096                                 return true;
08097                             })) {
08098         error = Error::Compression;
08099         return nullptr;
08100     }
08101 }
08102 } else
08103 #endif
08104 {
08105     if (content_provider) {
08106         req.content_length_ = content_length;
08107         req.content_provider_ = std::move(content_provider);
08108         req.is_chunked_content_provider_ = false;
08109     } else if (content_provider_without_length) {
08110         req.content_length_ = 0;
08111         req.content_provider_ = detail::ContentProviderAdapter(
08112             std::move(content_provider_without_length));
08113         req.is_chunked_content_provider_ = true;
08114         req.set_header("Transfer-Encoding", "chunked");
08115     } else {
08116         req.body.assign(body, content_length);
08117     }
08118 }
08119
08120 auto res = detail::make_unique<Response>();
08121 return send(req, *res, error) ? std::move(res) : nullptr;
08122 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.104 set_address_family()

```
void httplib::ClientImpl::set_address_family (
    int family) [inline]
```

См. определение в файле [httplib.h](#) строка 9002

```
09002
09003     address_family_ = family;
09004 }
```

6.5.3.105 `set_basic_auth()`

```
void httpplib::ClientImpl::set_basic_auth (
    const std::string & username,
    const std::string & password) [inline]
```

См. определение в файле [httpplib.h](#) строка 8964

```
08965 {
08966     basic_auth_username = username;
08967     basic_auth_password = password;
08968 }
```

6.5.3.106 `set_bearer_token_auth()`

```
void httpplib::ClientImpl::set_bearer_token_auth (
    const std::string & token) [inline]
```

См. определение в файле [httpplib.h](#) строка 8970

```
08970 {
08971     bearer_token_auth_token = token;
08972 }
```

6.5.3.107 `set_compress()`

```
void httpplib::ClientImpl::set_compress (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 9014

```
09014 { compress = on; }
```

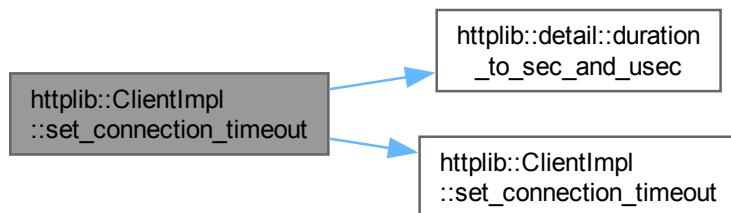
6.5.3.108 `set_connection_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::ClientImpl::set_connection_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2293

```
02294 {
02295     detail::duration_to_sec_and_usec(duration, [&](time_t sec, time_t usec) {
02296         set_connection_timeout(sec, usec);
02297     });
02298 }
```

Граф вызовов:



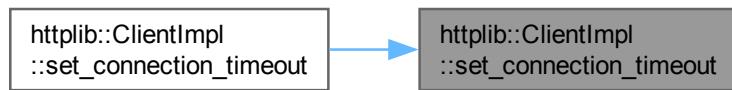
6.5.3.109 `set_connection_timeout()` [2/2]

```
void httpplib::ClientImpl::set_connection_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле `httpplib.h` строка 8945

```
08945 {
08946   connection_timeout_sec_ = sec;
08947   connection_timeout_usec_ = usec;
08948 }
```

Граф вызова функции:

6.5.3.110 `set_decompress()`

```
void httpplib::ClientImpl::set_decompress (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 9016

```
09016 { decompress_ = on; }
```

6.5.3.111 `set_default_headers()`

```
void httpplib::ClientImpl::set_default_headers (
    Headers headers) [inline]
```

См. определение в файле `httpplib.h` строка 8993

```
08993 {
08994   default_headers_ = std::move(headers);
08995 }
```

6.5.3.112 `set_follow_location()`

```
void httpplib::ClientImpl::set_follow_location (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 8984

```
08984 { follow_location_ = on; }
```

6.5.3.113 `set_header_writer()`

```
void httpplib::ClientImpl::set_header_writer (
    std::function<ssize_t(Stream &, Headers &)> const & writer) [inline]
```

См. определение в файле `httpplib.h` строка 8997

```
08998 {  
08999     header_writer_ = writer;  
09000 }
```

6.5.3.114 `set_hostname_addr_map()`

```
void httpplib::ClientImpl::set_hostname_addr_map (
    std::map<std::string, std::string> addr_map) [inline]
```

См. определение в файле `httpplib.h` строка 8989

```
08989 {  
08990     addr_map_ = std::move(addr_map);  
08991 }
```

6.5.3.115 `set_interface()`

```
void httpplib::ClientImpl::set_interface (
    const std::string & intf) [inline]
```

См. определение в файле `httpplib.h` строка 9018

```
09018 {  
09019     interface_ = intf;  
09020 }
```

6.5.3.116 `set_ipv6_v6only()`

```
void httpplib::ClientImpl::set_ipv6_v6only (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 9008

```
09008 { ipv6_v6only_ = on; }
```

6.5.3.117 `set_keep_alive()`

```
void httpplib::ClientImpl::set_keep_alive (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 8982

```
08982 { keep_alive_ = on; }
```

6.5.3.118 `set_logger()`

```
void httpplib::ClientImpl::set_logger (
    Logger logger) [inline]
```

См. определение в файле `httpplib.h` строка 9094

```
09094 {  
09095     logger_ = std::move(logger);  
09096 }
```

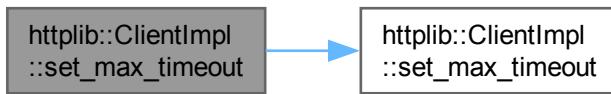
6.5.3.119 `set_max_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::ClientImpl::set_max_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2315

```
02316                                     {
02317     auto msec =
02318         std::chrono::duration_cast<std::chrono::milliseconds>(duration).count();
02319     set_max_timeout(msec);
02320 }
```

Граф вызовов:

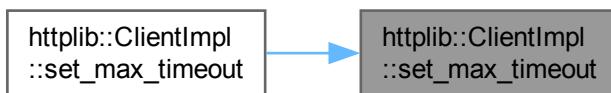
6.5.3.120 `set_max_timeout()` [2/2]

```
void httpplib::ClientImpl::set_max_timeout (
    time_t msec) [inline]
```

См. определение в файле [httpplib.h](#) строка 8960

```
08960                                     {
08961     max_timeout_msec_ = msec;
08962 }
```

Граф вызова функции:



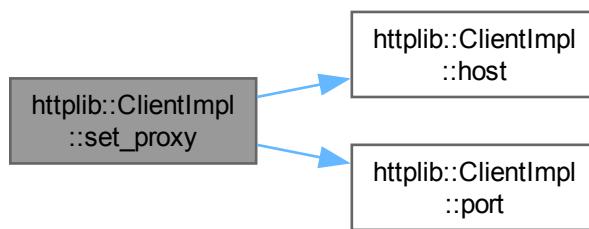
6.5.3.121 `set_proxy()`

```
void httpplib::ClientImpl::set_proxy (
    const std::string & host,
    int port) [inline]
```

См. определение в файле [httpplib.h](#) строка 9022

```
09022                                     {
09023     proxy_host_ = host;
09024     proxy_port_ = port;
09025 }
```

Граф вызовов:

6.5.3.122 `set_proxy_basic_auth()`

```
void httpplib::ClientImpl::set_proxy_basic_auth (
    const std::string & username,
    const std::string & password) [inline]
```

См. определение в файле [httpplib.h](#) строка 9027

```
09028                                     {
09029     proxy_basic_auth_username_ = username;
09030     proxy_basic_auth_password_ = password;
09031 }
```

6.5.3.123 `set_proxy_bearer_token_auth()`

```
void httpplib::ClientImpl::set_proxy_bearer_token_auth (
    const std::string & token) [inline]
```

См. определение в файле [httpplib.h](#) строка 9033

```
09033                                     {
09034     proxy_bearer_token_auth_token_ = token;
09035 }
```

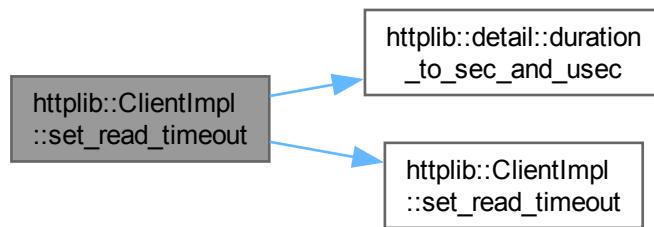
6.5.3.124 `set_read_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::ClientImpl::set_read_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2301

```
02302 {
02303     detail::duration_to_sec_and_usec(
02304         duration, [&](time_t sec, time_t usec) { set_read_timeout(sec, usec); });
02305 }
```

Граф вызовов:

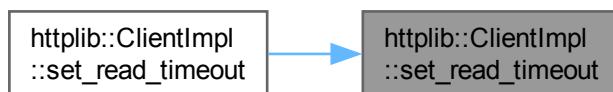
6.5.3.125 `set_read_timeout()` [2/2]

```
void httpplib::ClientImpl::set_read_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле [httpplib.h](#) строка 8950

```
08950 {
08951     read_timeout_sec_ = sec;
08952     read_timeout_usec_ = usec;
08953 }
```

Граф вызова функции:



6.5.3.126 `set_socket_options()`

```
void httpplib::ClientImpl::set_socket_options (
    SocketOptions socket_options) [inline]
```

См. определение в файле [httpplib.h](#) строка 9010

```
09010 {
09011     socket_options_ = std::move(socket_options);
09012 }
```

6.5.3.127 `set_tcp_nodelay()`

```
void httpplib::ClientImpl::set_tcp_nodelay (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 9006

```
09006 { tcp_nodelay_ = on; }
```

6.5.3.128 `set_url_encode()`

```
void httpplib::ClientImpl::set_url_encode (
    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 8986

```
08986 { url_encode_ = on; }
```

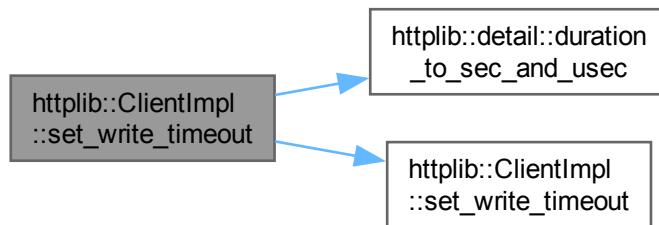
6.5.3.129 `set_write_timeout()` [1/2]

```
template<class Rep, class Period>
void httpplib::ClientImpl::set_write_timeout (
    const std::chrono::duration<Rep, Period> & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2308

```
02309 {
02310     detail::duration_to_sec_and_usec(
02311         duration, [&](time_t sec, time_t usec) { set_write_timeout(sec, usec); });
02312 }
```

Граф вызовов:



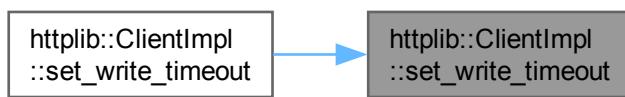
6.5.3.130 set_write_timeout() [2/2]

```
void httplib::ClientImpl::set_write_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

См. определение в файле [httplib.h](#) строка 8955

```
08955                                     {
08956     write_timeout_sec_ = sec;
08957     write_timeout_usec_ = usec;
08958 }
```

Граф вызова функции:



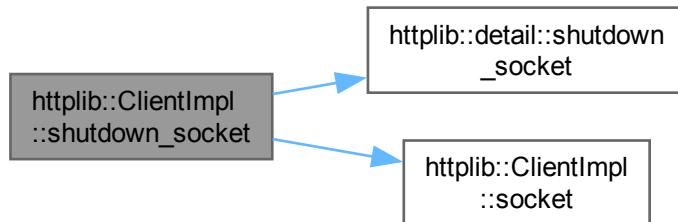
6.5.3.131 shutdown_socket()

```
void httplib::ClientImpl::shutdown_socket (
    Socket & socket) const [inline], [protected]
```

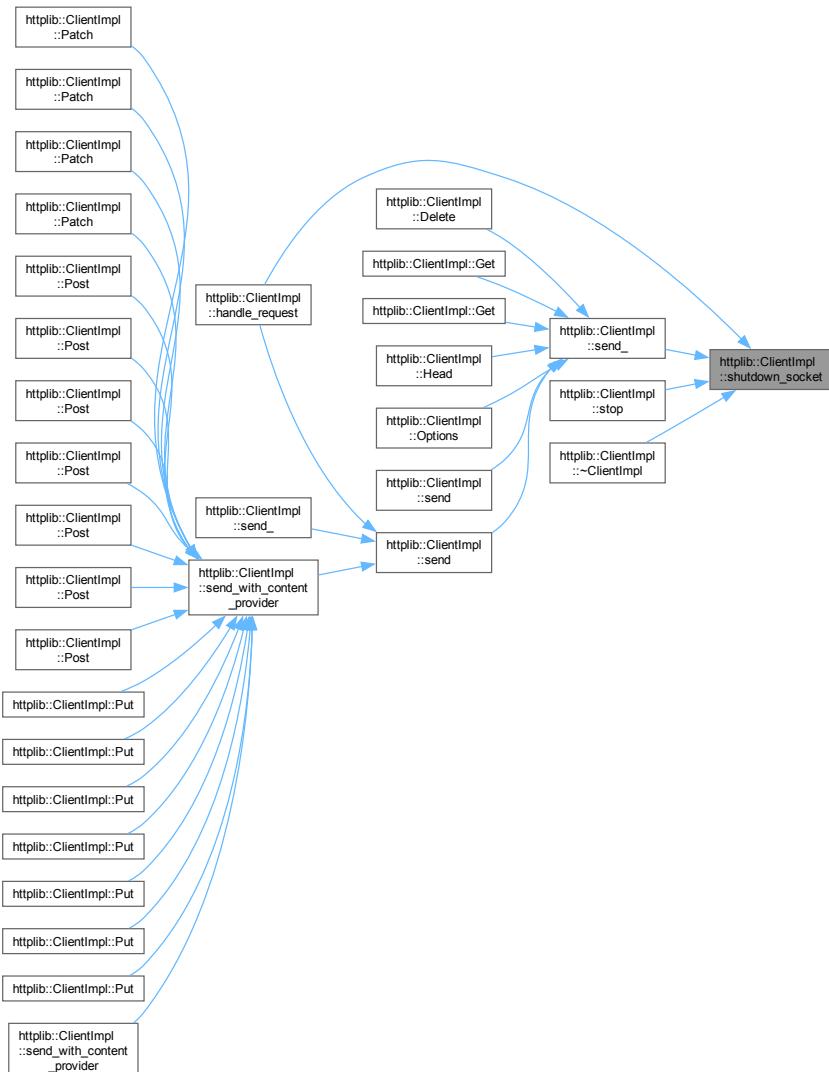
См. определение в файле [httplib.h](#) строка 7563

```
07563                                     {
07564     if (socket.sock == INVALID_SOCKET) { return; }
07565     detail::shutdown_socket(socket.sock);
07566 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.132 shutdown_ssl()

```

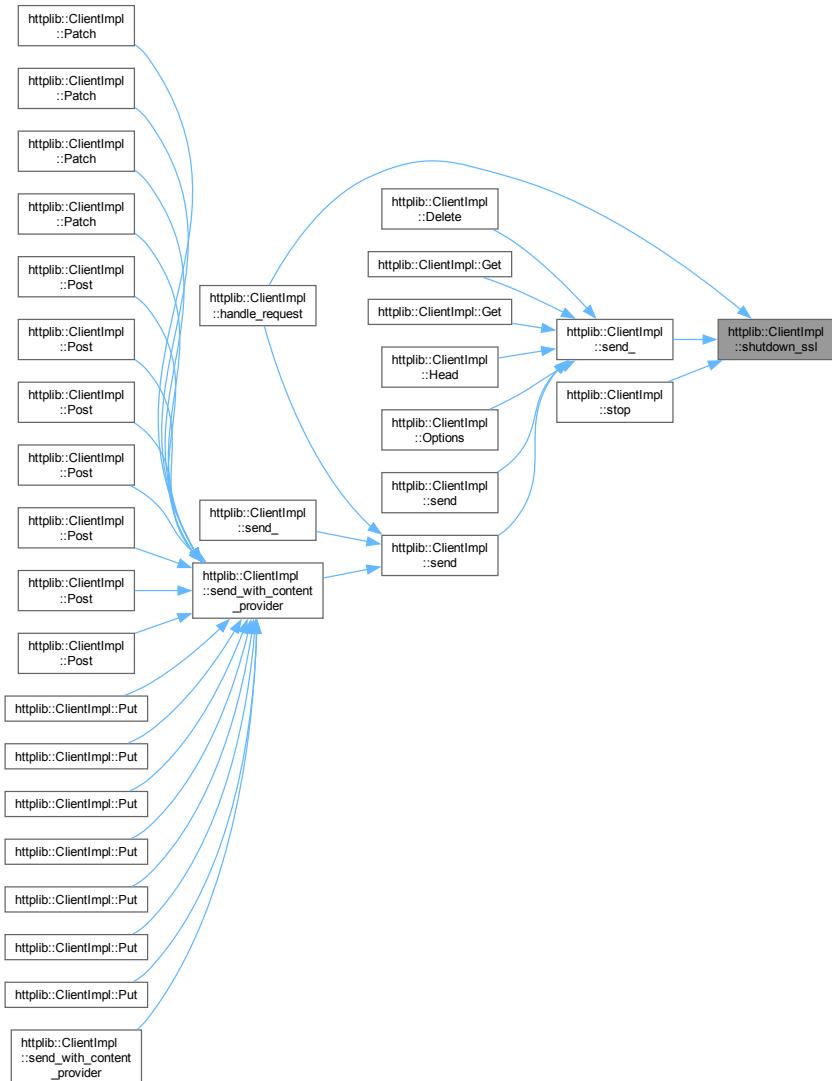
void httplib::ClientImpl::shutdown_ssl (
    Socket & socket,
    bool shutdown_gracefully) [inline], [protected], [virtual]
  
```

См. определение в файле `httplib.h` строка 7555

```

07556 {
07557 // If there are any requests in flight from threads other than us, then it's
07558 // a thread-unsafe race because individual ssl* objects are not thread-safe.
07559 assert(socket_requests_in_flight == 0 ||
07560     socket_requests_are_from_thread == std::this_thread::get_id());
07561 }
  
```

Граф вызова функции:



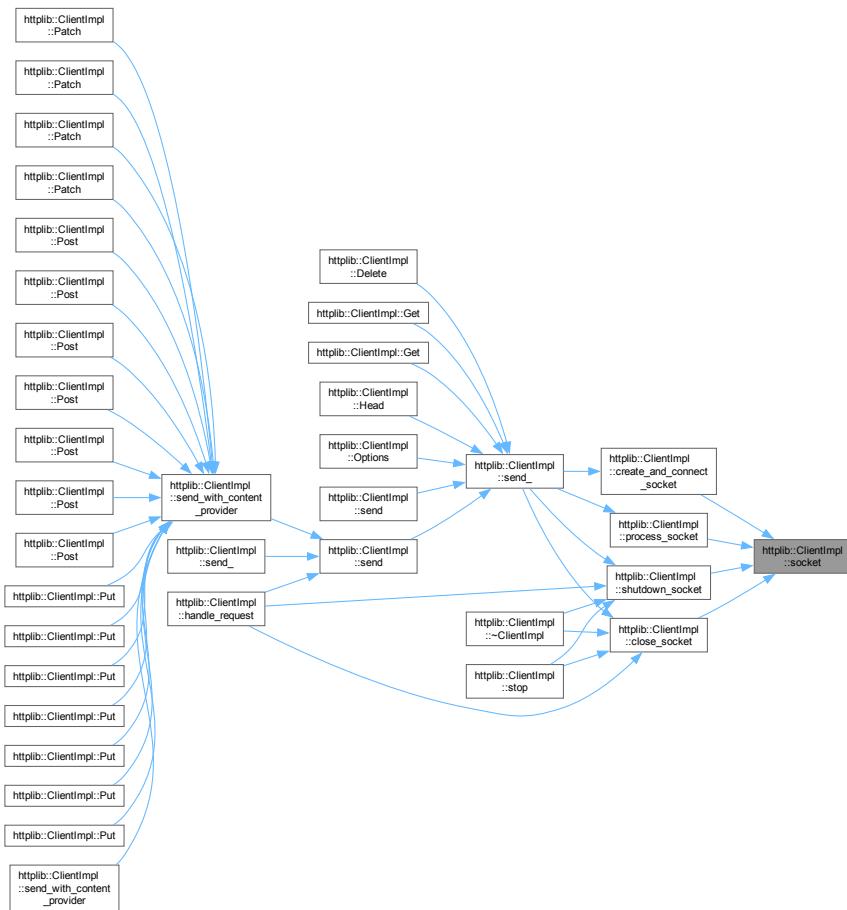
6.5.3.133 socket()

`socket_t` `httpplib::ClientImpl::socket () const [inline]`

См. определение в файле `httpplib.h` строка 8943

```
08943 { return socket_.sock; }
```

Граф вызова функции:



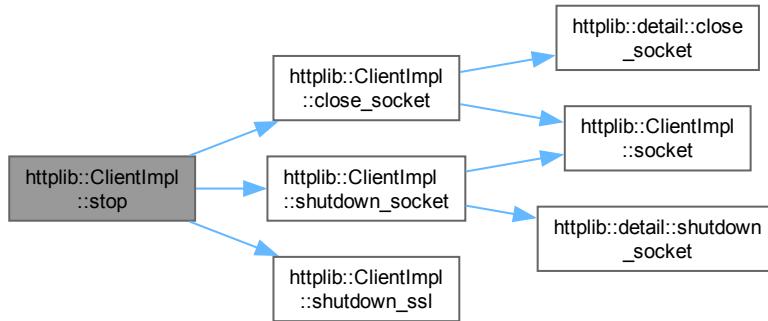
6.5.3.134 stop()

void httpplib::ClientImpl::stop () [inline]

См. определение в файле `httpplib.h` строка 8911

```
08911 {  
08912     std::lock_guard<std::mutex> guard(socket_mutex_);  
08913  
08914     // If there is anything ongoing right now, the ONLY thread-safe thing we can  
08915     // do is to shutdown_socket, so that threads using this socket suddenly  
08916     // discover they can't read/write any more and error out. Everything else  
08917     // (closing the socket, shutting ssl down) is unsafe because these actions are  
08918     // not thread-safe.  
08919     if (socket_requests_in_flight_ > 0) {  
08920         shutdown_socket(socket_);  
08921  
08922     // Aside from that, we set a flag for the socket to be closed when we're  
08923     // done.  
08924     socket_should_be_closed_when_request_is_done_ = true;  
08925     return;  
08926 }  
08927  
08928 // Otherwise, still holding the mutex, we can shut everything down ourselves  
08929 shutdown_ssl(socket_, true);  
08930 shutdown_socket(socket_);  
08931 close_socket(socket_);  
08932 }
```

Граф вызовов:



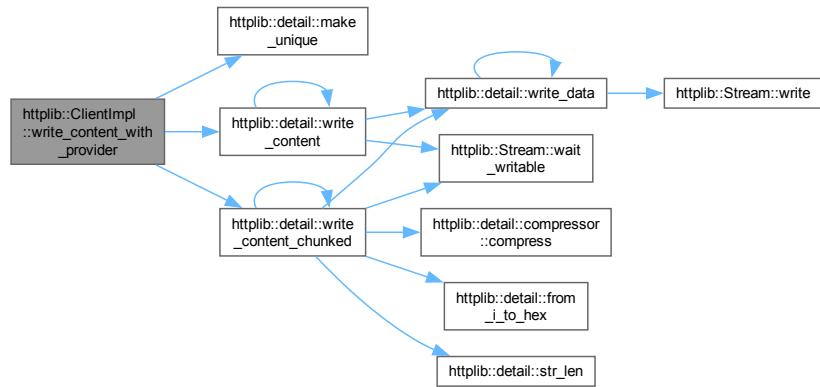
6.5.3.135 write_content_with_provider()

```
bool httplib::ClientImpl::write_content_with_provider (
    Stream & strm,
    const Request & req,
    Error & error) const [inline], [protected]
```

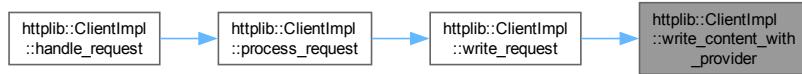
См. определение в файле [httplib.h](#) строка 7878

```
07880     {
07881     auto is_shutting_down = []() { return false; };
07882
07883     if (req.is_chunked_content_provider_) {
07884         // TODO: Brotli support
07885         std::unique_ptr<detail::compressor> compressor;
07886 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
07887         if (compress_) {
07888             compressor = detail::make_unique<detail::gzip_compressor>();
07889         } else
07890 #endif
07891         {
07892             compressor = detail::make_unique<detail::nocompressor>();
07893         }
07894
07895         return detail::write_content_chunked(strm, req.content_provider_,
07896                                              is_shutting_down, *compressor, error);
07897     } else {
07898         return detail::write_content(strm, req.content_provider_, 0,
07899                                     req.content_length_, is_shutting_down, error);
07900     }
07901 }
```

Граф вызовов:



Граф вызова функции:



6.5.3.136 write_request()

```

bool http://ClientImpl::write_request (
    Stream & strm,
    Request & req,
    bool close_connection,
    Error & error) [inline], [private]
  
```

См. определение в файле [http://ClientImpl.h](#) строка 7903

```

07904
07905 // Prepare additional headers
07906 if (close_connection) {
07907     if (!req.has_header("Connection")) {
07908         req.set_header("Connection", "close");
07909     }
07910 }
07911
07912 if (!req.has_header("Host")) {
07913     if (is_ssl()) {
07914         if (port_ == 443) {
07915             req.set_header("Host", host_);
07916         } else {
07917             req.set_header("Host", host_and_port_);
07918         }
07919     } else {
07920         if (port_ == 80) {
07921             req.set_header("Host", host_);
07922         } else {
07923             req.set_header("Host", host_and_port_);
07924         }
07925     }
07926 }
  
```

```

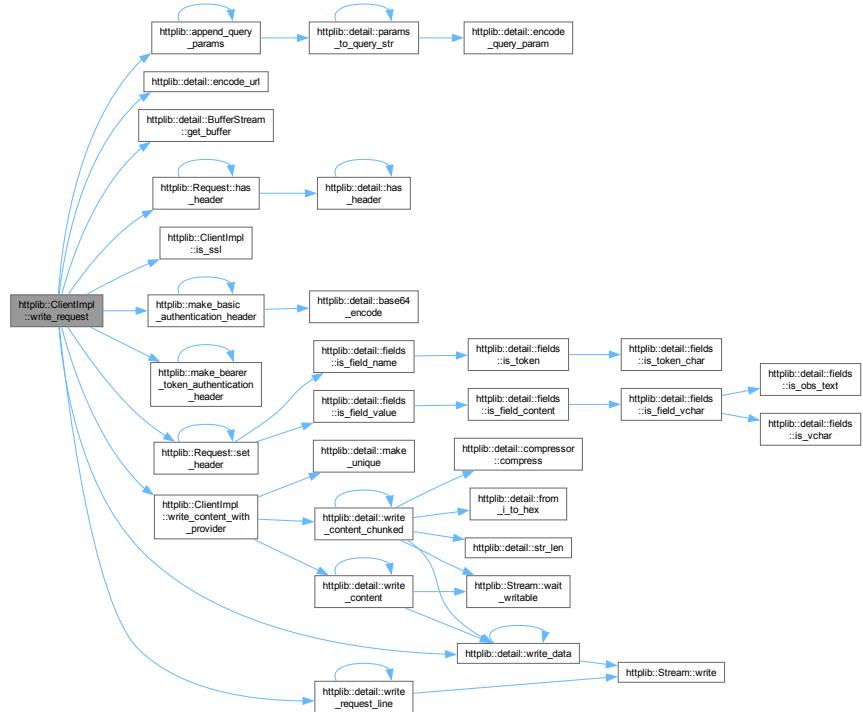
07927
07928 if (!req.has_header("Accept")) { req.set_header("Accept", "*/*"); }
07929
07930 if (!req.content_receiver) {
07931   if (!req.has_header("Accept-Encoding")) {
07932     std::string accept_encoding;
07933 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
07934   accept_encoding = "br";
07935 #endif
07936 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
07937   if (!accept_encoding.empty()) { accept_encoding += ", "; }
07938   accept_encoding += "gzip, deflate";
07939 #endif
07940 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
07941   if (!accept_encoding.empty()) { accept_encoding += ", "; }
07942   accept_encoding += "zstd";
07943 #endif
07944   req.set_header("Accept-Encoding", accept_encoding);
07945 }
07946
07947 #ifndef CPPHTTPLIB_NO_DEFAULT_USER_AGENT
07948   if (!req.has_header("User-Agent")) {
07949     auto agent = std::string("cpp-httplib/") + CPPHTTPLIB_VERSION;
07950     req.set_header("User-Agent", agent);
07951   }
07952 #endif
07953 };
07954
07955 if (req.body.empty()) {
07956   if (req.content_provider_) {
07957     if (!req.is_chunked_content_provider_) {
07958       if (!req.has_header("Content-Length")) {
07959         auto length = std::to_string(req.content_length_);
07960         req.set_header("Content-Length", length);
07961     }
07962   }
07963 } else {
07964   if (req.method == "POST" || req.method == "PUT" ||
07965       req.method == "PATCH") {
07966     req.set_header("Content-Length", "0");
07967   }
07968 }
07969 } else {
07970   if (!req.has_header("Content-Type")) {
07971     req.set_header("Content-Type", "text/plain");
07972   }
07973
07974 if (!req.has_header("Content-Length")) {
07975   auto length = std::to_string(req.body.size());
07976   req.set_header("Content-Length", length);
07977 }
07978 }
07979
07980 if (!basic_auth_password_.empty() || !basic_auth_username_.empty()) {
07981   if (!req.has_header("Authorization")) {
07982     req.headers.insert(make_basic_authentication_header(
07983       basic_auth_username_, basic_auth_password_, false));
07984   }
07985 }
07986
07987 if (!proxy_basic_auth_username_.empty() &&
07988   !proxy_basic_auth_password_.empty()) {
07989   if (!req.has_header("Proxy-Authorization")) {
07990     req.headers.insert(make_basic_authentication_header(
07991       proxy_basic_auth_username_, proxy_basic_auth_password_, true));
07992   }
07993 }
07994
07995 if (!bearer_token_auth_token_.empty()) {
07996   if (!req.has_header("Authorization")) {
07997     req.headers.insert(make_bearer_token_authentication_header(
07998       bearer_token_auth_token_, false));
07999   }
08000 }
08001
08002 if (!proxy_bearer_token_auth_token_.empty()) {
08003   if (!req.has_header("Proxy-Authorization")) {
08004     req.headers.insert(make_bearer_token_authentication_header(
08005       proxy_bearer_token_auth_token_, true));
08006   }
08007 }
08008
08009 // Request line and headers
08010 {
08011   detail::BufferStream bstrm;
08012
08013   const auto &path_with_query =

```

```

08014     req.params.empty() ? req.path
08015         : append_query_params(req.path, req.params);
08016
08017     const auto &path =
08018         url_encode_ ? detail::encode_url(path_with_query) : path_with_query;
08019
08020     detail::write_request_line(bstrm, req.method, path);
08021
08022     header_writer_(bstrm, req.headers);
08023
08024     // Flush buffer
08025     auto &data = bstrm.get_buffer();
08026     if (!detail::write_data(strm, data.data(), data.size())) {
08027         error = Error::Write;
08028         return false;
08029     }
08030 }
08031
08032     // Body
08033     if (req.body.empty()) {
08034         return write_content_with_provider(strm, req, error);
08035     }
08036
08037     if (!detail::write_data(strm, req.body.data(), req.body.size())) {
08038         error = Error::Write;
08039         return false;
08040     }
08041
08042     return true;
08043 }
```

Граф вызовов:



Граф вызова функции:



6.5.4 Данные класса

6.5.4.1 `addr_map_`

```
std::map<std::string, std::string> httpplib::ClientImpl::addr_map_ [protected]
```

См. определение в файле [httpplib.h](#) строка 1548

6.5.4.2 `address_family_`

```
int httpplib::ClientImpl::address_family_ = AF_UNSPEC [protected]
```

См. определение в файле [httpplib.h](#) строка 1582

6.5.4.3 `basic_auth_password_`

```
std::string httpplib::ClientImpl::basic_auth_password_ [protected]
```

См. определение в файле [httpplib.h](#) строка 1570

6.5.4.4 `basic_auth_username_`

```
std::string httpplib::ClientImpl::basic_auth_username_ [protected]
```

См. определение в файле [httpplib.h](#) строка 1569

6.5.4.5 `bearer_token_auth_token_`

```
std::string httpplib::ClientImpl::bearer_token_auth_token_ [protected]
```

См. определение в файле [httpplib.h](#) строка 1571

6.5.4.6 `client_cert_path_`

```
std::string httpplib::ClientImpl::client_cert_path_ [protected]
```

См. определение в файле [httpplib.h](#) строка 1558

6.5.4.7 `client_key_path_`

`std::string httpplib::ClientImpl::client_key_path_ [protected]`

См. определение в файле [httpplib.h](#) строка [1559](#)

6.5.4.8 `compress_`

`bool httpplib::ClientImpl::compress_ = false [protected]`

См. определение в файле [httpplib.h](#) строка [1587](#)

6.5.4.9 `connection_timeout_sec_`

`time_t httpplib::ClientImpl::connection_timeout_sec_ = 300 [protected]`

См. определение в файле [httpplib.h](#) строка [1561](#)

6.5.4.10 `connection_timeout_usec_`

`time_t httpplib::ClientImpl::connection_timeout_usec_ = 0 [protected]`

См. определение в файле [httpplib.h](#) строка [1562](#)

6.5.4.11 `decompress_`

`bool httpplib::ClientImpl::decompress_ = true [protected]`

См. определение в файле [httpplib.h](#) строка [1588](#)

6.5.4.12 `default_headers_`

`Headers httpplib::ClientImpl::default_headers_ [protected]`

См. определение в файле [httpplib.h](#) строка [1551](#)

6.5.4.13 `follow_location_`

`bool httpplib::ClientImpl::follow_location_ = false [protected]`

См. определение в файле [httpplib.h](#) строка [1578](#)

6.5.4.14 `header_writer_`

```
std::function<ssize_t(Stream &, Headers &)> httpplib::ClientImpl::header_writer_ [protected]
```

Инициализатор

```
= detail::write_headers
```

См. определение в файле [httpplib.h](#) строка [1554](#)

6.5.4.15 `host_`

```
const std::string httpplib::ClientImpl::host_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1533](#)

6.5.4.16 `host_and_port_`

```
const std::string httpplib::ClientImpl::host_and_port_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1535](#)

6.5.4.17 `interface_`

```
std::string httpplib::ClientImpl::interface_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1590](#)

6.5.4.18 `ipv6_v6only_`

```
bool httpplib::ClientImpl::ipv6_v6only_ = false [protected]
```

См. определение в файле [httpplib.h](#) строка [1584](#)

6.5.4.19 `keep_alive_`

```
bool httpplib::ClientImpl::keep_alive_ = false [protected]
```

См. определение в файле [httpplib.h](#) строка [1577](#)

6.5.4.20 `logger_`

```
Logger httpplib::ClientImpl::logger_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1616](#)

6.5.4.21 `max_timeout_msec_`

```
time_t httpplib::ClientImpl::max_timeout_msec_ = 0 [protected]
```

См. определение в файле [httpplib.h](#) строка [1567](#)

6.5.4.22 `port_`

```
const int httpplib::ClientImpl::port_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1534](#)

6.5.4.23 `proxy_basic_auth_password_`

```
std::string httpplib::ClientImpl::proxy_basic_auth_password_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1596](#)

6.5.4.24 `proxy_basic_auth_username_`

```
std::string httpplib::ClientImpl::proxy_basic_auth_username_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1595](#)

6.5.4.25 `proxy_bearer_token_auth_token_`

```
std::string httpplib::ClientImpl::proxy_bearer_token_auth_token_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1597](#)

6.5.4.26 `proxy_host_`

```
std::string httpplib::ClientImpl::proxy_host_ [protected]
```

См. определение в файле [httpplib.h](#) строка [1592](#)

6.5.4.27 `proxy_port_`

```
int httpplib::ClientImpl::proxy_port_ = -1 [protected]
```

См. определение в файле [httpplib.h](#) строка [1593](#)

6.5.4.28 `read_timeout_sec_`

```
time_t httpplib::ClientImpl::read_timeout_sec_ = 300 [protected]
```

См. определение в файле [httpplib.h](#) строка [1563](#)

6.5.4.29 `read_timeout_usec_`

`time_t httpplib::ClientImpl::read_timeout_usec_ = 0` [protected]

См. определение в файле [httpplib.h](#) строка 1564

6.5.4.30 `request_mutex_`

`std::recursive_mutex httpplib::ClientImpl::request_mutex_` [protected]

См. определение в файле [httpplib.h](#) строка 1540

6.5.4.31 `socket_`

`Socket httpplib::ClientImpl::socket_` [protected]

См. определение в файле [httpplib.h](#) строка 1538

6.5.4.32 `socket_mutex_`

`std::mutex httpplib::ClientImpl::socket_mutex_` [mutable], [protected]

См. определение в файле [httpplib.h](#) строка 1539

6.5.4.33 `socket_options_`

`SocketOptions httpplib::ClientImpl::socket_options_ = nullptr` [protected]

См. определение в файле [httpplib.h](#) строка 1585

6.5.4.34 `socket_requests_are_from_thread_`

`std::thread::id httpplib::ClientImpl::socket_requests_are_from_thread_ = std::thread::id()` [protected]

См. определение в файле [httpplib.h](#) строка 1544

6.5.4.35 `socket_requests_in_flight_`

`size_t httpplib::ClientImpl::socket_requests_in_flight_ = 0` [protected]

См. определение в файле [httpplib.h](#) строка 1543

6.5.4.36 `socket_should_be_closed_when_request_is_done_`

`bool httpplib::ClientImpl::socket_should_be_closed_when_request_is_done_ = false` [protected]

См. определение в файле [httpplib.h](#) строка 1545

6.5.4.37 `tcp_nodelay_`

```
bool httpplib::ClientImpl::tcp_nodelay_ = false [protected]
```

См. определение в файле [httpplib.h](#) строка 1583

6.5.4.38 `url_encode_`

```
bool httpplib::ClientImpl::url_encode_ = true [protected]
```

См. определение в файле [httpplib.h](#) строка 1580

6.5.4.39 `write_timeout_sec_`

```
time_t httpplib::ClientImpl::write_timeout_sec_ = 5 [protected]
```

См. определение в файле [httpplib.h](#) строка 1565

6.5.4.40 `write_timeout_usec_`

```
time_t httpplib::ClientImpl::write_timeout_usec_ = 0 [protected]
```

См. определение в файле [httpplib.h](#) строка 1566

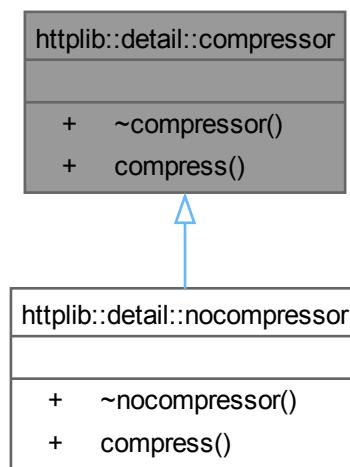
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

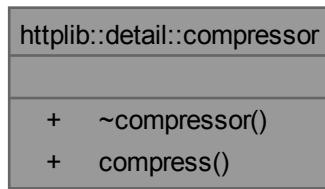
6.6 Класс `httpplib::detail::compressor`

```
#include <httpplib.h>
```

Граф наследования:`httpplib::detail::compressor`:



Граф связей класса `httpplib::detail::compressor`:



Открытые типы

- `typedef std::function< bool(const char *data, size_t data_len)> Callback`

Открытые члены

- `virtual ~compressor ()=default`
- `virtual bool compress (const char *data, size_t data_length, bool last, Callback callback)=0`

6.6.1 Подробное описание

См. определение в файле [httpplib.h](#) строка [2481](#)

6.6.2 Определения типов

6.6.2.1 Callback

`typedef std::function<bool(const char *data, size_t data_len)> httpplib::detail::compressor::Callback`

См. определение в файле [httpplib.h](#) строка [2485](#)

6.6.3 Конструктор(ы)

6.6.3.1 `~compressor()`

`virtual httpplib::detail::compressor::~compressor () [virtual], [default]`

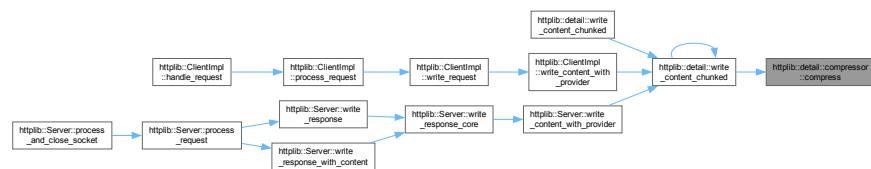
6.6.4 Методы

6.6.4.1 `compress()`

```
virtual bool httpplib::detail::compressor::compress (
    const char * data,
    size_t data_length,
    bool last,
    Callback callback) [pure virtual]
```

Замещается в [httpplib::detail::nocompressor](#).

Граф вызова функции:



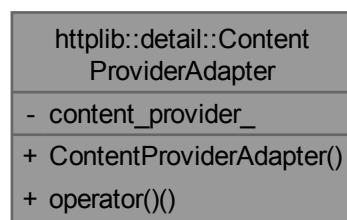
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.7 Класс `httpplib::detail::ContentProviderAdapter`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::ContentProviderAdapter`:



Открытые члены

- `ContentProviderAdapter (ContentProviderWithoutLength &&content_provider)`
- `bool operator() (size_t offset, size_t, DataSink &sink)`

Закрытые данные

- `ContentProviderWithoutLength content_provider_`

6.7.1 Подробное описание

См. определение в файле `httpplib.h` строка 5764

6.7.2 Конструктор(ы)

6.7.2.1 `ContentProviderAdapter()`

```
httpplib::detail::ContentProviderAdapter::ContentProviderAdapter (
    ContentProviderWithoutLength && content_provider) [inline], [explicit]
```

См. определение в файле `httpplib.h` строка 5766

```
05768     : content_provider_(content_provider) {}
```

6.7.3 Методы

6.7.3.1 `operator()()`

```
bool httpplib::detail::ContentProviderAdapter::operator() (
    size_t offset,
    size_t ,
    DataSink & sink) [inline]
```

См. определение в файле `httpplib.h` строка 5770

```
05770
05771     return content_provider_(offset, sink);
05772 }
```

6.7.4 Данные класса

6.7.4.1 `content_provider_`

`ContentProviderWithoutLength httpplib::detail::ContentProviderAdapter::content_provider_` [private]

См. определение в файле `httpplib.h` строка 5775

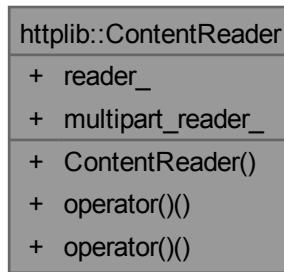
Объявления и описания членов класса находятся в файле:

- `httpplib.h`

6.8 Класс `httpplib::ContentReader`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::ContentReader`:



Открытые типы

- using `Reader` = `std::function<bool(ContentReceiver receiver)>`
- using `MultipartReader`

Открытые члены

- `ContentReader (Reader reader, MultipartReader multipart_reader)`
- `bool operator() (MultipartContentHeader header, ContentReceiver receiver) const`
- `bool operator() (ContentReceiver receiver) const`

Открытые атрибуты

- `Reader reader_`
- `MultipartReader multipart_reader_`

6.8.1 Подробное описание

См. определение в файле `httpplib.h` строка 601

6.8.2 Определения типов

6.8.2.1 MultipartReader

```
using httpplib::ContentReader::MultipartReader
```

Инициализатор

```
std::function<bool(MultipartContentHeader header,  
ContentReceiver receiver)>
```

См. определение в файле `httpplib.h` строка 604

6.8.2.2 Reader

```
using httpplib::ContentReader::Reader = std::function<bool(ContentReceiver receiver)>
```

См. определение в файле `httpplib.h` строка 603

6.8.3 Конструктор(ы)

6.8.3.1 ContentReader()

```
httpplib::ContentReader::ContentReader (
    Reader reader,
    MultipartReader multipart_reader) [inline]
```

См. определение в файле `httpplib.h` строка 607

```
00608     : reader_(std::move(reader)),
00609     multipart_reader_(std::move(multipart_reader)) {}
```

6.8.4 Методы

6.8.4.1 operator()() [1/2]

```
bool httpplib::ContentReader::operator() (
    ContentReceiver receiver) const [inline]
```

См. определение в файле `httpplib.h` строка 616

```
00616 {
00617     return reader_(std::move(receiver));
00618 }
```

6.8.4.2 operator()() [2/2]

```
bool httpplib::ContentReader::operator() (
    MultipartContentHeader header,
    ContentReceiver receiver) const [inline]
```

См. определение в файле `httpplib.h` строка 611

```
00612 {
00613     return multipart_reader_(std::move(header), std::move(receiver));
00614 }
```

6.8.5 Данные класса

6.8.5.1 multipart_reader_

`MultipartReader` `httpplib::ContentReader::multipart_reader_`

См. определение в файле `httpplib.h` строка 621

6.8.5.2 `reader_`

`Reader` `httpplib::ContentReader::reader_`

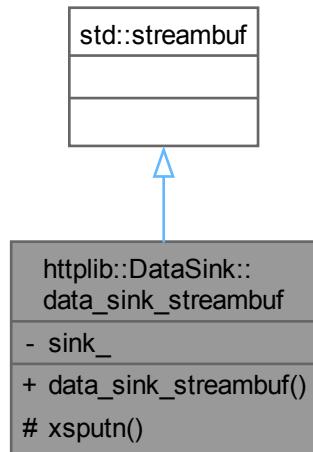
См. определение в файле `httpplib.h` строка 620

Объявления и описания членов класса находятся в файле:

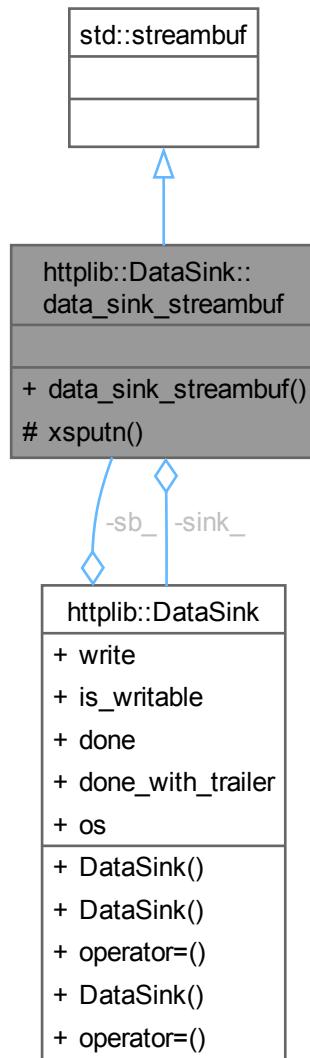
- `httpplib.h`

6.9 Класс `httpplib::DataSink::data_sink_streambuf`

Граф наследования: `httpplib::DataSink::data_sink_streambuf`:



Граф связей класса `httpplib::DataSink::data_sink_streambuf`:



Открытые члены

- `data_sink_streambuf (DataSink & sink)`

Запущенные члены

- `std::streamsize xsputn (const char *s, std::streamsize n) override`

Закрытые данные

- `DataSink & sink_`

6.9.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 558

6.9.2 Конструктор(ы)

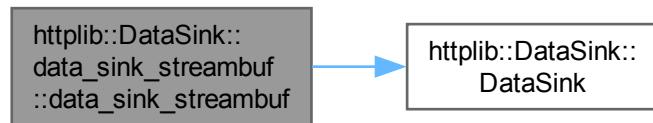
6.9.2.1 `data_sink_streambuf()`

```
httpplib::DataSink::data_sink_streambuf::data_sink_streambuf (
    DataSink & sink) [inline], [explicit]
```

См. определение в файле [httpplib.h](#) строка 560

```
00560 : sink_(sink) {}
```

Граф вызовов:



6.9.3 Методы

6.9.3.1 `xsputn()`

```
std::streamsize httpplib::DataSink::data_sink_streambuf::xsputn (
    const char * s,
    std::streamsize n) [inline], [override], [protected]
```

См. определение в файле [httpplib.h](#) строка 563

```
00563     {
00564         sink_.write(s, static_cast<size_t>(n));
00565         return n;
00566     }
```

6.9.4 Данные класса

6.9.4.1 `sink_`

`DataSink&` `httpplib::DataSink::data_sink_streambuf::sink_` [private]

См. определение в файле [httpplib.h](#) строка 569

Объявления и описания членов класса находятся в файле:

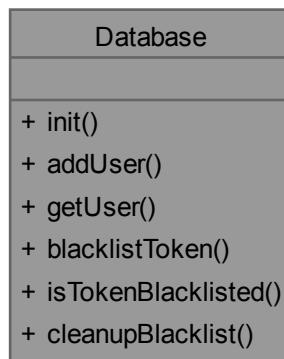
- [httpplib.h](#)

6.10 Класс Database

Класс-обёртка для работы с SQLite-базой данных.

```
#include <Database.h>
```

Граф связей класса Database:



Открытые статические члены

- static bool **init** (const std::string &db_path)
Инициализирует подключение к SQLite-базе данных.
- static bool **addUser** (const std::string &username, const std::string &password)
Добавляет нового пользователя в таблицу users.
- static bool **getUser** (const std::string &username, User &user_out)
Получает пользователя из базы данных по имени.
- static bool **blacklistToken** (const std::string &token, uint64_t expires_at)
Добавляет токен в таблицу blacklist (например, при logout).
- static bool **isTokenBlacklisted** (const std::string &token)
Проверяет, содержится ли токен в blacklist.
- static bool **cleanupBlacklist** ()
Удаляет все устаревшие токены из blacklist (истёкшие по времени).

6.10.1 Подробное описание

Класс-обёртка для работы с SQLite-базой данных.

Обеспечивает доступ к таблице пользователей и таблице заблокированных (blacklisted) токенов. Используется для реализации регистрации, авторизации, выхода из системы и защиты от повторного использования refresh токенов.

См. определение в файле [Database.h](#) строка 20

6.10.2 Методы

6.10.2.1 addUser()

```
bool Database::addUser (
    const std::string & username,
    const std::string & password) [static]
```

Добавляет нового пользователя в таблицу users.

Перед вставкой хеширует пароль.

Аргументы

username	Имя пользователя.
password	Обычный текстовый пароль (будет захеширован).

Возвращает

true, если пользователь успешно добавлен; false, если уже существует или произошла ошибка.

См. определение в файле [Database.cpp](#) строка 51

```
00051           {
00052     std::string hashed = PasswordEncryptor::hashPassword(password);
00053     const char* sql = "INSERT INTO users (username, password) VALUES (?, ?);";
00054
00055     sqlite3_stmt* stmt;
00056     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00057
00058     sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_TRANSIENT);
00059     sqlite3_bind_text(stmt, 2, hashed.c_str(), -1, SQLITE_TRANSIENT);
00060
00061     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00062     sqlite3_finalize(stmt);
00063     return success;
00064 }
```

Граф вызовов:



Граф вызова функции:



6.10.2.2 blacklistToken()

```
bool Database::blacklistToken (
    const std::string & token,
    uint64_t expires_at) [static]
```

Добавляет токен в таблицу blacklist (например, при logout).

Используется для блокировки refresh токена до момента его истечения.

Аргументы

token	Строковое представление JWT токена.
expires_at	Метка времени, когда токен истекает (UNIX-время).

Возвращает

true, если вставка прошла успешно или токен уже есть; false — в случае ошибки.

См. определение в файле [Database.cpp](#) строка 90

```
00090     const char* sql = "INSERT OR IGNORE INTO blacklist (token, expires_at) VALUES (?, ?);";
00091     sqlite3_stmt* stmt;
00092     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00093     sqlite3_bind_text(stmt, 1, token.c_str(), -1, SQLITE_TRANSIENT);
00094     sqlite3_bind_int64(stmt, 2, static_cast<sqlite3_int64>(expires_at));
00095     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00096     sqlite3_finalize(stmt);
00097     return success;
00098 }
00099 }
```

Граф вызова функции:



6.10.2.3 cleanupBlacklist()

```
bool Database::cleanupBlacklist () [static]
```

Удаляет все устаревшие токены из blacklist (истёкшие по времени).

Вызывается периодически для очистки таблицы и экономии ресурсов.

Возвращает

true, если удаление прошло успешно; false — при ошибке выполнения запроса.

См. определение в файле [Database.cpp](#) строка 115

```
00115     {
00116     const char* sql = "DELETE FROM blacklist WHERE expires_at < ?;";
00117     sqlite3_stmt* stmt;
00118     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00119
00120     sqlite3_bind_int64(stmt, 1, static_cast<sqlite3_int64>(std::time(nullptr)));
00121
00122     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00123     sqlite3_finalize(stmt);
00124     return success;
00125 }
```

6.10.2.4 getUser()

```
bool Database::getUser (
    const std::string & username,
    User & user_out) [static]
```

Получает пользователя из базы данных по имени.

Если пользователь найден, возвращает его через user_out.

Аргументы

username	Имя пользователя.
user_out	Объект, в который будут загружены данные.

Возвращает

true, если пользователь найден; false в противном случае.

См. определение в файле [Database.cpp](#) строка 66

```
00066     {
00067     const char* sql = "SELECT id, username, password FROM users WHERE username = ?;";
00068     sqlite3_stmt* stmt;
00069     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00070
00071     sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_TRANSIENT);
00072
00073     bool found = false;
00074     int rc = sqlite3_step(stmt);
00075     if (rc == SQLITE_ROW) {
00076         user_out.id = sqlite3_column_int(stmt, 0);
00077         user_out.username = (const char*)sqlite3_column_text(stmt, 1);
00078         user_out.password = (const char*)sqlite3_column_text(stmt, 2);
00079         found = true;
00080     }
00081
00082     sqlite3_finalize(stmt);
00083     return found;
00084 }
```

Граф вызова функции:



6.10.2.5 init()

```
bool Database::init (
    const std::string & db_path) [static]
```

Инициализирует подключение к SQLite-базе данных.

Создаёт при необходимости две таблицы:

- users(id, username, password)
- blacklist(token, expires_at)

Аргументы

db_path	Путь к файлу базы данных.
---------	---------------------------

Возвращает

true, если инициализация прошла успешно; false в случае ошибки.

См. определение в файле [Database.cpp](#) строка 9

```
00009     {
00010     int rc = sqlite3_open(db_path.c_str(), &db);
00011     if (rc) {
00012         std::cerr << "Can't open database: " << sqlite3_errmsg(db) << "\n";
00013         return false;
00014     }
00015
00016     const char* create_users_sql = R"(
00017         CREATE TABLE IF NOT EXISTS users (
00018             id INTEGER PRIMARY KEY AUTOINCREMENT,
00019             username TEXT UNIQUE NOT NULL,
00020             password TEXT NOT NULL
00021         );
00022     )";
00023
00024     const char* create_blacklist_sql = R"(
00025         CREATE TABLE IF NOT EXISTS blacklist (
00026             token TEXT PRIMARY KEY,
00027             expires_at INTEGER NOT NULL
00028         );
00029     )";
00030
00031     char* errMsg = nullptr;
00032
00033     rc = sqlite3_exec(db, create_users_sql, nullptr, nullptr, &errMsg);
00034     if (rc != SQLITE_OK) {
00035         std::cerr << "SQL error (users): " << errMsg << "\n";
00036         sqlite3_free(errMsg);
00037         return false;
00038     }
00039
00040     rc = sqlite3_exec(db, create_blacklist_sql, nullptr, nullptr, &errMsg);
00041     if (rc != SQLITE_OK) {
00042         std::cerr << "SQL error (blacklist): " << errMsg << "\n";
00043         sqlite3_free(errMsg);
00044         return false;
00045     }
00046
00047     std::cout << "Database initialized successfully.\n";
00048     return true;
00049 }
```

Граф вызова функции:



6.10.2.6 isTokenBlacklisted()

```
bool Database::isTokenBlacklisted (
    const std::string & token) [static]
```

Проверяет, содержится ли токен в blacklist.

Вызывается, чтобы убедиться, что refresh токен не был отозван.

Аргументы

token	Проверяемый JWT токен.
-------	------------------------

Возвращает

true, если токен есть в blacklist; false, если его нет.

См. определение в файле [Database.cpp](#) строка 103

```
00103     {
00104     const char* sql = "SELECT 1 FROM blacklist WHERE token = ? LIMIT 1;";
00105     sqlite3_stmt* stmt;
00106     if(sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00107
00108     sqlite3_bind_text(stmt, 1, token.c_str(), -1, SQLITE_TRANSIENT);
00109
00110     bool found = (sqlite3_step(stmt) == SQLITE_ROW);
00111     sqlite3_finalize(stmt);
00112     return found;
00113 }
```

Граф вызова функции:



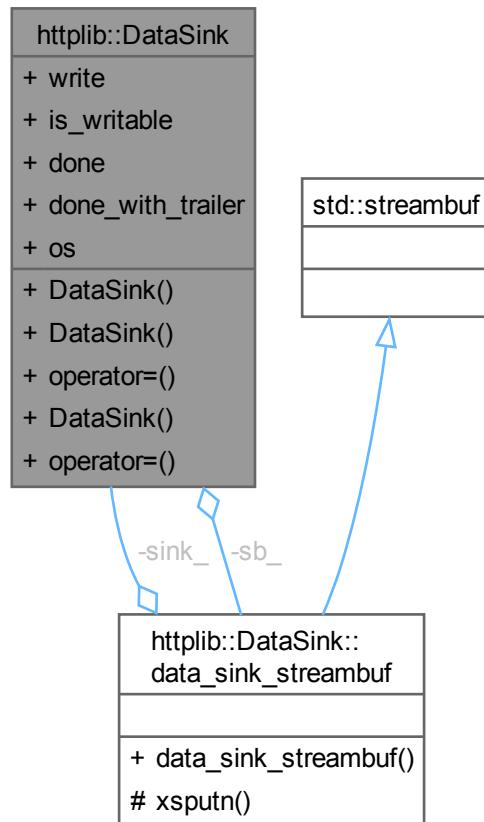
Объявления и описания членов классов находятся в файлах:

- [Database.h](#)
- [Database.cpp](#)

6.11 Класс httpplib::DataSink

```
#include <httpplib.h>
```

Граф связей класса httpplib::DataSink:



Классы

- class [data_sink_streambuf](#)

Открытые члены

- `DataSink ()`
- `DataSink (const DataSink &) = delete`
- `DataSink & operator= (const DataSink &) = delete`
- `DataSink (DataSink &&) = delete`
- `DataSink & operator= (DataSink &&) = delete`

Открытые атрибуты

- `std::function< bool(const char *data, size_t data_len)> write`
- `std::function< bool()> is_writable`
- `std::function< void()> done`
- `std::function< void(const Headers &trailer)> done_with_trailer`
- `std::ostream os`

Закрытые данные

- `data_sink_streambuf sb_`

6.11.1 Подробное описание

См. определение в файле `httpplib.h` строка [542](#)

6.11.2 Конструктор(ы)

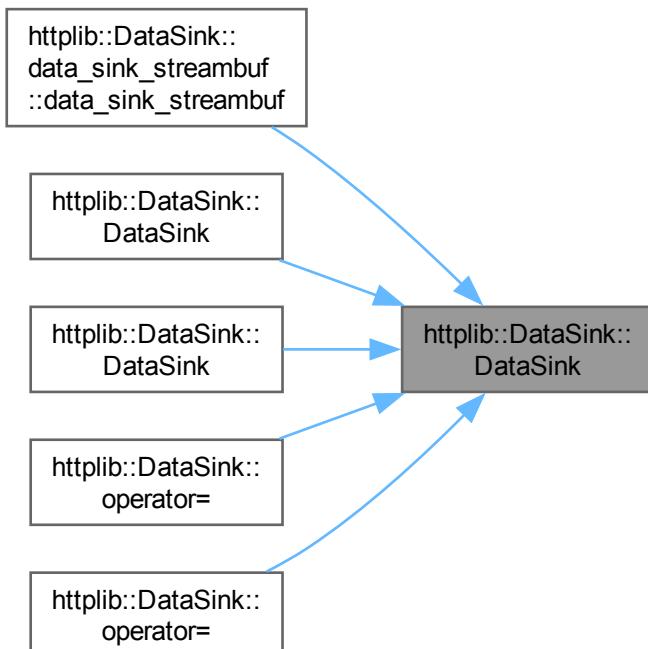
6.11.2.1 `DataSink()` [1/3]

`httpplib::DataSink::DataSink () [inline]`

См. определение в файле `httpplib.h` строка [544](#)

`00544 : os(&sb_), sb_(*this) {}`

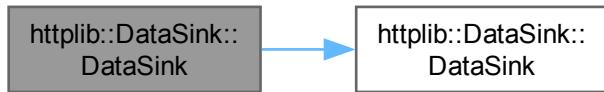
Граф вызова функции:



6.11.2.2 DataSink() [2/3]

```
httplib::DataSink::DataSink (const DataSink & ) [delete]
```

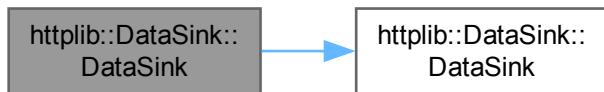
Граф вызовов:



6.11.2.3 DataSink() [3/3]

```
httplib::DataSink::DataSink (DataSink && ) [delete]
```

Граф вызовов:

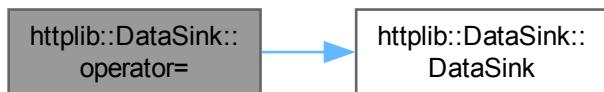


6.11.3 Методы

6.11.3.1 operator=() [1/2]

```
DataSink & httplib::DataSink::operator= (const DataSink & ) [delete]
```

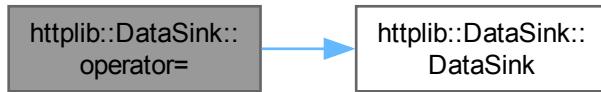
Граф вызовов:



6.11.3.2 `operator=()` [2/2]

```
DataSink & httpplib::DataSink::operator= (
    DataSink && ) [delete]
```

Граф вызовов:



6.11.4 Данные класса

6.11.4.1 `done`

```
std::function<void()> httpplib::DataSink::done
```

См. определение в файле [httpplib.h](#) строка [553](#)

6.11.4.2 `done_with_trailer`

```
std::function<void(const Headers &trailer)> httpplib::DataSink::done_with_trailer
```

См. определение в файле [httpplib.h](#) строка [554](#)

6.11.4.3 `is_writable`

```
std::function<bool()> httpplib::DataSink::is_writable
```

См. определение в файле [httpplib.h](#) строка [552](#)

6.11.4.4 `os`

```
std::ostream httpplib::DataSink::os
```

См. определение в файле [httpplib.h](#) строка [555](#)

6.11.4.5 `sb_`

```
data_sink_streambuf httpplib::DataSink::sb_ [private]
```

См. определение в файле [httpplib.h](#) строка [572](#)

6.11.4.6 write

`std::function<bool(const char *data, size_t data_len)> httpplib::DataSink::write`

См. определение в файле [httpplib.h](#) строка 551

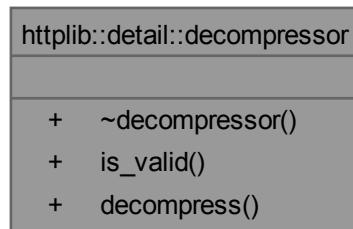
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.12 Класс `httpplib::detail::decompressor`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::decompressor`:



Открытые типы

- `typedef std::function< bool(const char *data, size_t data_len)> Callback`

Открытые члены

- `virtual ~decompressor ()=default`
- `virtual bool is_valid () const =0`
- `virtual bool decompress (const char *data, size_t data_length, Callback callback)=0`

6.12.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 2490

6.12.2 Определения типов

6.12.2.1 Callback

```
typedef std::function<bool(const char *data, size_t data_len)> httpplib::detail::decompressor::Callback
```

См. определение в файле `httpplib.h` строка 2496

6.12.3 Конструктор(ы)

6.12.3.1 ~decompressor()

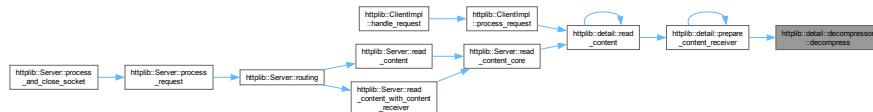
```
virtual httpplib::detail::decompressor::~decompressor () [virtual], [default]
```

6.12.4 Методы

6.12.4.1 decompress()

```
virtual bool httpplib::detail::decompressor::decompress (
    const char * data,
    size_t data_length,
    Callback callback) [pure virtual]
```

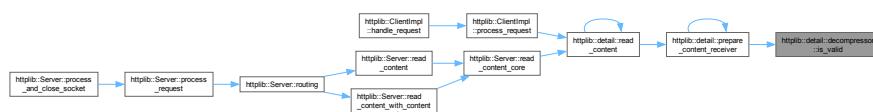
Граф вызова функции:



6.12.4.2 is_valid()

```
virtual bool httpplib::detail::decompressor::is_valid () const [pure virtual]
```

Граф вызова функции:



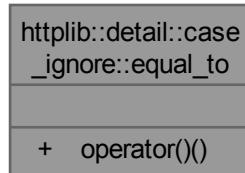
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.13 Структура `httpplib::detail::case_ignore::equal_to`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::case_ignore::equal_to`:



Открытые члены

- `bool operator()(const std::string &a, const std::string &b) const`

6.13.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 383

6.13.2 Методы

6.13.2.1 `operator()()`

```
bool httpplib::detail::case_ignore::equal_to::operator() (
    const std::string & a,
    const std::string & b) const [inline]
```

См. определение в файле [httpplib.h](#) строка 384

```
00384
00385     return equal(a, b);
00386 }
```

Граф вызовов:



Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.14 Структура `httpplib::detail::FileStat`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::FileStat`:

httpplib::detail::FileStat	
-	<code>st_</code>
-	<code>ret_</code>
+	<code>FileStat()</code>
+	<code>is_file()</code>
+	<code>is_dir()</code>

Открытые члены

- `FileStat` (const std::string &path)
- bool `is_file` () const
- bool `is_dir` () const

Закрытые данные

- struct stat `st_`
- int `ret_` = -1

6.14.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 2382

6.14.2 Конструктор(ы)

6.14.2.1 `FileStat()`

```
httpplib::detail::FileStat::FileStat (
    const std::string & path) [inline]
```

См. определение в файле [httpplib.h](#) строка 2850

```
02850
02851 #if defined(_WIN32)
02852     auto wpath = u8string_to_wstring(path.c_str());
02853     ret_ = _wstat(wpath.c_str(), &st_);
02854 #else
02855     ret_ = stat(path.c_str(), &st_);
02856 #endif
02857 }
```

6.14.3 Методы

6.14.3.1 `is_dir()`

`bool httpplib::detail::FileStat::is_dir () const [inline]`

См. определение в файле [httpplib.h](#) строка [2861](#)

```
02861
02862     return ret_ >= 0 && S_ISDIR(st_.st_mode);
02863 }
```

Граф вызова функции:



6.14.3.2 `is_file()`

`bool httpplib::detail::FileStat::is_file () const [inline]`

См. определение в файле [httpplib.h](#) строка [2858](#)

```
02858
02859     return ret_ >= 0 && S_ISREG(st_.st_mode);
02860 }
```

Граф вызова функции:



6.14.4 Данные класса

6.14.4.1 `ret_`

`int httpplib::detail::FileStat::ret_ = -1 [private]`

См. определение в файле [httpplib.h](#) строка [2393](#)

6.14.4.2 `st_`

`struct stat httpplib::detail::FileStat::st_ [private]`

См. определение в файле [httpplib.h](#) строка [2391](#)

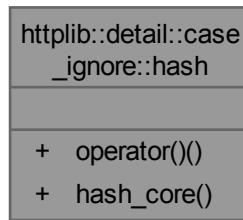
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.15 Структура `httpplib::detail::case_ignore::hash`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::case_ignore::hash`:



Открытые члены

- `size_t operator()` (`const std::string &key`) const
- `size_t hash_core` (`const char *s, size_t l, size_t h`) const

6.15.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 389

6.15.2 Методы

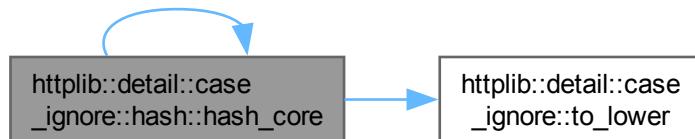
6.15.2.1 `hash_core()`

```
size_t httpplib::detail::case_ignore::hash::hash_core (
    const char * s,
    size_t l,
    size_t h) const [inline]
```

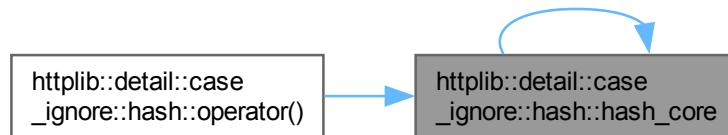
См. определение в файле [httpplib.h](#) строка 394

```
00394     return (l == 0) ? h
00395     : hash_core(s + 1, l - 1,
00396                  // Unsets the 6 high bits of h, therefore no
00397                  // overflow happens
00398                  (((std::numeric_limits<size_t>::max()) >> 6) &
00399                  h * 33) ^
00400                  static_cast<unsigned char>(to_lower(*s)));
00401
00402 }
```

Граф вызовов:



Граф вызова функции:



6.15.2.2 operator()()

```
size_t httpplib::detail::case_ignore::hash::operator() (
    const std::string & key) const [inline]
```

См. определение в файле [httpplib.h](#) строка 390

```
00390     {
00391     return hash_core(key.data(), key.size(), 0);
00392 }
```

Граф вызовов:



Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.16 Класс HttpServer

Класс для запуска HTTP-сервера авторизации.

```
#include <HttpServer.h>
```

Граф связей класса HttpServer:



Открытые статические члены

- static void `start` (int port=8080)
Запускает HTTP-сервер на заданном порту.

6.16.1 Подробное описание

Класс для запуска HTTP-сервера авторизации.

Этот класс реализует REST API сервер с маршрутами:

- POST /register — регистрация нового пользователя.
- POST /login — аутентификация пользователя, выдача access и refresh токенов.
- POST /refresh — обновление access токена по действующему refresh токену.
- GET /secure/data — доступ к защищённым данным по access токену.
- POST /logout — добавление refresh токена в blacklist.

Сервер построен на базе библиотеки cpp-httplib и поддерживает CORS.

См. определение в файле [HttpServer.h](#) строка 16

6.16.2 Методы

6.16.2.1 start()

```
void HttpServer::start (
    int port = 8080) [static]
```

Запускает HTTP-сервер на заданном порту.

Все маршруты, связанные с авторизацией, автоматически настраиваются внутри функции. Также активируется логгирование запросов и включается поддержка CORS.

Аргументы

port	Порт, на котором будет слушать сервер (по умолчанию: 8080).
------	-------------------------------------------------------------

См. определение в файле [HttpServer.cpp](#) строка 29

```

00029     {
00030     httplib::Server server;
00031
00032     server.set_logger([](const httplib::Request& req, const httplib::Response& res) {
00033         std::cout << "[LOGGER] " << req.method << " " << req.path << " -> " << res.status << "\n";
00034     });
00035
00036     server.Options(R"(.*)", [](const httplib::Request&, httplib::Response& res) {
00037         res.set_header("Access-Control-Allow-Origin", "*");
00038         res.set_header("Access-Control-Allow-Methods", "POST, GET, OPTIONS");
00039         res.set_header("Access-Control-Allow-Headers", "Content-Type, Authorization");
00040         res.status = 200;
00041     });
00042
00043     server.Post("/register", [](const httplib::Request& req, httplib::Response& res) {
00044         res.set_header("Access-Control-Allow-Origin", "*");
00045         std::cout << "[REGISTER] Получен запрос: " << req.body << std::endl;
00046
00047         std::string username = extractField(req.body, "username");
00048         std::string password = extractField(req.body, "password");
00049
00050         if (username.empty() || password.empty()) {
00051             std::cerr << "[REGISTER] Отсутствует username или password" << std::endl;
00052             res.status = 400;
00053             res.set_content("Missing 'username' or 'password'", "text/plain");
00054             return;
00055         }
00056
00057         std::cout << "[REGISTER] Имя пользователя: " << username << std::endl;
00058
00059         if (!Database::addUser(username, password)) {
00060             std::cerr << "[REGISTER] Пользователь уже существует" << std::endl;
00061             res.status = 409;
00062             res.set_content("Username already exists", "text/plain");
00063             return;
00064         }
00065
00066         std::cout << "[REGISTER] Регистрация успешна" << std::endl;
00067         res.status = 201;
00068         res.set_content("User registered successfully", "text/plain");
00069     });
00070
00071     server.Post("/login", [](const httplib::Request& req, httplib::Response& res) {
00072         res.set_header("Access-Control-Allow-Origin", "*");
00073         std::cout << "[LOGIN] Получен запрос: " << req.body << std::endl;
00074
00075         std::string username = extractField(req.body, "username");
00076         std::string password = extractField(req.body, "password");
00077
00078         if (username.empty() || password.empty()) {
00079             std::cerr << "[LOGIN] Отсутствует username или password" << std::endl;
00080             res.status = 400;
00081             res.set_content("Missing 'username' or 'password'", "text/plain");
00082             return;
00083         }
00084
00085         std::cout << "[LOGIN] Имя пользователя: " << username << std::endl;
00086
00087         User user;
00088         if (!Database::getUser(username, user)) {
00089             std::cerr << "[LOGIN] Пользователь не найден в базе данных" << std::endl;
00090             res.status = 401;
00091             res.set_content("Invalid credentials", "text/plain");
00092             return;
00093         }
00094
00095         std::string hashedInput = PasswordEncryptor::hashPassword(password);
00096         std::cout << "[LOGIN] Введённый пароль (хеш): " << hashedInput << std::endl;
00097         std::cout << "[LOGIN] Хеш пароля из базы: " << user.password << std::endl;
00098
00099         if (hashedInput != user.password) {
00100             std::cerr << "[LOGIN] Неверный пароль" << std::endl;
00101             res.status = 401;
00102             res.set_content("Invalid credentials", "text/plain");
00103             return;
00104         }
00105
00106         RSAPublicKey pubKey;

```

```

00107     RSAPrivatekey privKey;
00108     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00109         std::cerr << "[LOGIN] Ошибка загрузки ключей" << std::endl;
00110         res.status = 500;
00111         res.set_content("Key error", "text/plain");
00112         return;
00113     }
00114
00115     std::string accessToken = JWT::createAccessToken(username, 60 * 1, privKey); // 1 минута
00116     std::string refreshToken = JWT::createRefreshToken(username, 60 * 60, privKey); // 60 минут
00117
00118     std::cout << "[LOGIN] Сгенерирован Access токен:" << accessToken << std::endl;
00119     std::cout << "[LOGIN] Сгенерирован Refresh токен:" << refreshToken << std::endl;
00120
00121     std::string response = "{}";
00122     response += "\"access_token\":\"" + accessToken + "\",";
00123     response += "\"refresh_token\":\"" + refreshToken + "\",";
00124     response += "}";
00125
00126     res.set_content(response, "application/json");
00127     std::cout << "[LOGIN] Ответ отправлен клиенту\n" << std::endl;
00128 });
00129
00130 server.Post("/refresh", [] (const httpplib::Request& req, httpplib::Response& res) {
00131     res.set_header("Access-Control-Allow-Origin", "*");
00132     std::cout << "\n[SERVER] --- /refresh endpoint called ---\n";
00133
00134     if (!req.has_header("Authorization")) {
00135         std::cerr << "[ERROR] Missing Authorization header\n";
00136         res.status = 400;
00137         res.set_content("Missing Authorization header", "text/plain");
00138         return;
00139     }
00140
00141     std::string authHeader = req.get_header_value("Authorization");
00142     std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00143
00144     std::string prefix = "Bearer ";
00145     if (authHeader.rfind(prefix, 0) != 0) {
00146         std::cerr << "[ERROR] Authorization header must start with 'Bearer '\n";
00147         res.status = 400;
00148         res.set_content("Invalid Authorization header format", "text/plain");
00149         return;
00150     }
00151
00152     std::string refreshToken = authHeader.substr(prefix.size());
00153     std::cout << "[PARSE] Extracted refresh token: " << refreshToken << "\n";
00154
00155     RSApublickey pubKey;
00156     RSAprivatekey privKey;
00157     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00158         std::cerr << "[ERROR] Failed to load RSA keys from storage\n";
00159         res.status = 500;
00160         res.set_content("Key error", "text/plain");
00161         return;
00162     }
00163
00164     std::string username;
00165     std::cout << "[VERIFY] Verifying refresh token...\n";
00166     if (!JWT::verifyRefreshToken(refreshToken, pubKey, username)) {
00167         std::cerr << "[ERROR] Invalid or expired refresh token\n";
00168         res.status = 401;
00169         res.set_content("Invalid or expired refresh token", "text/plain");
00170         return;
00171     }
00172
00173     std::cout << "[JWT] Refresh token is valid.\n";
00174     std::cout << "[JWT] Extracted subject (username): " << username << "\n";
00175
00176     if (Database::isTokenBlacklisted(refreshToken)) {
00177         std::cerr << "[SECURITY] Refresh token is blacklisted. Rejected.\n";
00178         res.status = 403;
00179         res.set_content("Refresh token is blacklisted", "text/plain");
00180         return;
00181     }
00182
00183     std::cout << "[JWT] Token is not in blacklist. Proceeding to generate new access token...\n";
00184
00185     std::string newAccessToken = JWT::createAccessToken(username, 60, privKey); // 1 минута
00186     std::cout << "[JWT] New access token generated:\n" << newAccessToken << "\n";
00187
00188     std::string response = "{}";
00189     response += "\"access_token\":\"" + newAccessToken + "\",";
00190     response += "}";
00191
00192     std::cout << "[RESPONSE] JSON: " << response << "\n";
00193     std::cout << "[SERVER] --- /refresh complete ---\n";

```

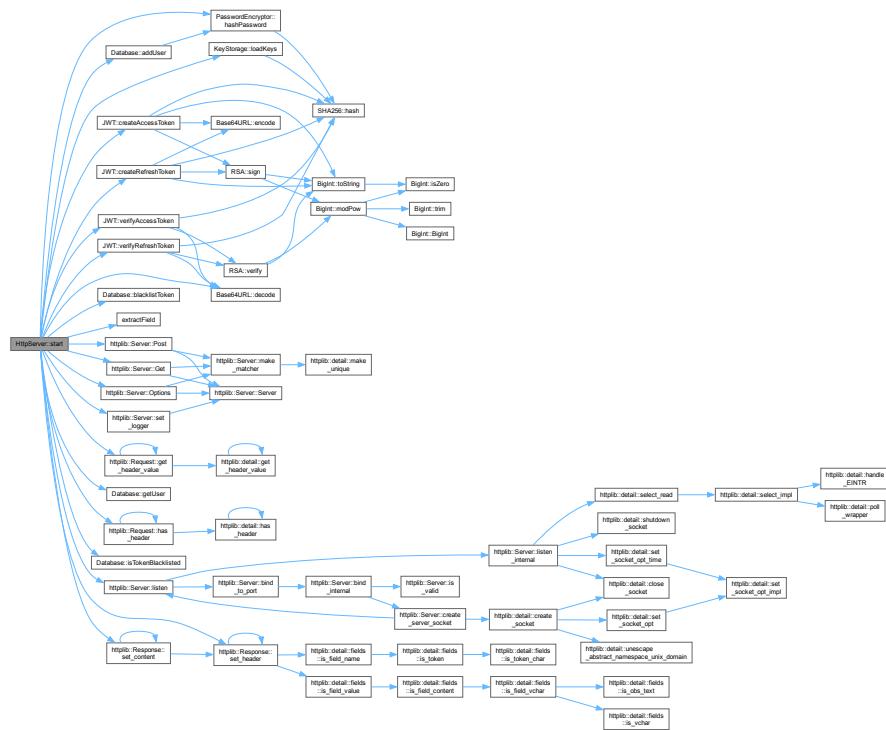
```

00194     res.set_content(response, "application/json");
00195   });
00196
00197
00198 server.Get("/secure/data", [](const httplib::Request& req, httplib::Response& res) {
00199   res.set_header("Access-Control-Allow-Origin", "*");
00200   std::cout << "\n[SERVER] --- /secure/data endpoint called ---\n";
00201
00202   // [1] Проверяем заголовок Authorization
00203   auto authHeaderIt = req.headers.find("Authorization");
00204   if (authHeaderIt == req.headers.end()) {
00205     std::cerr << "[ERROR] Missing 'Authorization' header\n";
00206     res.status = 401;
00207     res.set_content("Missing 'Authorization' header", "text/plain");
00208     return;
00209   }
00210
00211   std::string authHeader = authHeaderIt->second;
00212   std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00213
00214   if (authHeader.find("Bearer ") != 0) {
00215     std::cerr << "[ERROR] Invalid Authorization format (should start with 'Bearer ')\n";
00216     res.status = 400;
00217     res.set_content("Invalid Authorization format", "text/plain");
00218     return;
00219   }
00220
00221   std::string accessToken = authHeader.substr(7);
00222   std::cout << "[TOKEN] Extracted access token: " << accessToken << "\n";
00223
00224   // [2] Загружаем ключи
00225   RSApublicKey pubKey;
00226   RSAprivateKey privKey; // не нужен здесь, но оставим на случай доработок
00227   if (!KeyStorage::loadKeys(pubKey, privKey)) {
00228     std::cerr << "[ERROR] Failed to load RSA keys\n";
00229     res.status = 500;
00230     res.set_content("Key error", "text/plain");
00231     return;
00232   }
00233
00234   // [3] Проверяем токен
00235   std::string subject;
00236   std::cout << "[VERIFY] Verifying access token...\n";
00237   if (!JWT::verifyAccessToken(accessToken, pubKey, subject)) {
00238     std::cerr << "[ERROR] Invalid or expired access token\n";
00239     res.status = 401;
00240     res.set_content("Invalid or expired access token", "text/plain");
00241     return;
00242   }
00243
00244   std::cout << "[JWT] Access token is valid.\n";
00245   std::cout << "[JWT] Extracted subject (username): " << subject << "\n";
00246
00247   // [4] Возвращаем защищенные данные
00248   std::string secureData = "{ \"data\": \"Secret message for " + subject + "\" }";
00249   std::cout << "[RESPONSE] Sending secure data: " << secureData << "\n";
00250   std::cout << "[SERVER] --- /secure/data complete ---\n";
00251
00252   res.set_content(secureData, "application/json");
00253 });
00254
00255 server.Post("/logout", [](const httplib::Request& req, httplib::Response& res) {
00256   res.set_header("Access-Control-Allow-Origin", "*");
00257   std::cout << "\n[SERVER] --- /logout endpoint called ---\n";
00258
00259   if (!req.has_header("Authorization")) {
00260     std::cerr << "[ERROR] Missing Authorization header\n";
00261     res.status = 400;
00262     res.set_content("Missing Authorization header", "text/plain");
00263     return;
00264   }
00265
00266   std::string authHeader = req.get_header_value("Authorization");
00267   std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00268
00269   std::string prefix = "Bearer ";
00270   if (authHeader.rfind(prefix, 0) != 0) {
00271     std::cerr << "[ERROR] Authorization header must start with 'Bearer '\n";
00272     res.status = 400;
00273     res.set_content("Invalid Authorization header format", "text/plain");
00274     return;
00275   }
00276
00277   std::string refreshToken = authHeader.substr(prefix.size());
00278   std::cout << "[INPUT] Extracted refresh token: " << refreshToken << "\n";
00279
00280   RSApublicKey pubKey;

```

```
00281     RSAPrivateKey privKey;
00282     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00283         std::cerr << "[ERROR] Failed to load keys\n";
00284         res.status = 500;
00285         res.set_content("Key error", "text/plain");
00286         return;
00287     }
00288
00289     std::string subject;
00290     std::cout << "[VERIFY] Verifying refresh token...\n";
00291
00292     if (!JWT::verifyRefreshToken(refreshToken, pubKey, subject)) {
00293         std::cerr << "[ERROR] Invalid or expired refresh token\n";
00294         res.status = 401;
00295         res.set_content("Invalid or expired refresh token", "text/plain");
00296         return;
00297     }
00298
00299     std::cout << "[JWT] Token is valid. Subject: " << subject << "\n";
00300
00301 // ===== Вытаскиваем expires_at из payload =====
00302 std::string payloadB64 = refreshToken.substr(
00303     refreshToken.find('.') + 1,
00304     refreshToken.rfind('.') - refreshToken.find('.') - 1
00305 );
00306 std::string payloadJson = Base64URL::decode(payloadB64);
00307
00308 size_t expPos = payloadJson.find("\\" exp \":");
00309 if (expPos == std::string::npos) {
00310     std::cerr << "[ERROR] Cannot extract exp from token\n";
00311     res.status = 400;
00312     res.set_content("Invalid token payload", "text/plain");
00313     return;
00314 }
00315
00316 expPos += 6;
00317 size_t expEnd = payloadJson.find_first_of(",}", expPos);
00318 uint64_t expTime = std::stoull(payloadJson.substr(expPos, expEnd - expPos));
00319
00320 std::cout << "[BLACKLIST] Extracted exp time: " << expTime << "\n";
00321
00322 if (!Database::blacklistToken(refreshToken, expTime)) {
00323     std::cerr << "[ERROR] Failed to blacklist token\n";
00324     res.status = 500;
00325     res.set_content("Database error", "text/plain");
00326     return;
00327 }
00328
00329 std::cout << "[BLACKLIST] Token successfully blacklisted\n";
00330 std::cout << "[SERVER] --- /logout completed ---\n";
00331
00332     res.set_content("Logged out successfully", "text/plain");
00333 });
00334
00335 std::cout << "[HttpServer] Сервер запущен на порту " << port << std::endl;
00336 server.listen("0.0.0.0", port);
00337 }
```

Граф вызовов:



Граф вызова функции:



Объявления и описания членов классов находятся в файлах:

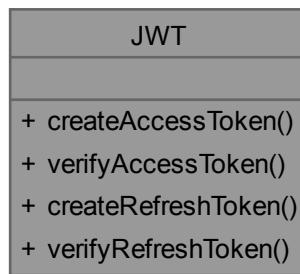
- [HttpServer.h](#)
- [HttpServer.cpp](#)

6.17 Класс JWT

Класс, реализующий создание и проверку JSON Web Token ([JWT](#)).

```
#include <JWT.h>
```

Граф связей класса JWT:



Открытые статические члены

- static std::string [createAccessToken](#) (const std::string &subject, uint64_t expirationSeconds, const [RSAPrivateKey](#) &privKey)

Создаёт access-токен для указанного пользователя.
- static bool [verifyAccessToken](#) (const std::string &token, const [RSAPublicKey](#) &pubKey, std::string &outSubject)

Проверяет access-токен на подлинность и срок действия.
- static std::string [createRefreshToken](#) (const std::string &subject, uint64_t expirationSeconds, const [RSAPrivateKey](#) &privKey)

Создаёт refresh-токен для указанного пользователя.
- static bool [verifyRefreshToken](#) (const std::string &token, const [RSAPublicKey](#) &pubKey, std::string &outSubject)

Проверяет refresh-токен на подлинность и срок действия.

6.17.1 Подробное описание

Класс, реализующий создание и проверку JSON Web Token ([JWT](#)).

[JWT](#) — это компактный URL-безопасный формат, используемый для представления утверждений (claims) между двумя сторонами. Здесь реализована поддержка токенов с алгоритмом RS256 ([RSA + SHA256](#)).

Структура токена:

- Header: информация об алгоритме подписи и типе ("alg": "RS256", "typ": "JWT")
- Payload: полезные данные, включая:
 - sub — subject (имя пользователя),
 - iat — время создания (issued at),
 - exp — время истечения,
 - typ — тип токена (access или refresh)
- Signature: RSA-подпись от (Base64(header) + "." + Base64(payload))

См. определение в файле [JWT.h](#) строка 21

6.17.2 Методы

6.17.2.1 createAccessToken()

```
std::string JWT::createAccessToken (
    const std::string & subject,
    uint64_t expirationSeconds,
    const RSAPrivateKey & privKey) [static]
```

Создаёт access-токен для указанного пользователя.

Алгоритм:

1. Формируется JSON header: `{"alg": "RS256", "typ": "JWT"}`
2. Формируется JSON payload с subject (имя пользователя), временем создания (iat), временем истечения (exp), и типом токена "access".
3. Оба JSON-объекта кодируются в [Base64URL](#).
4. Выполняется хеширование [SHA256](#) от соединённой строки: header.payload
5. Хеш подписывается приватным RSA-ключом.
6. Подпись также кодируется в [Base64URL](#) и добавляется к [JWT](#).

Аргументы

<code>subject</code>	Имя пользователя, для которого создаётся токен
<code>expirationSeconds</code>	Время жизни токена (в секундах)
<code>privKey</code>	Приватный RSA-ключ для подписи

Возвращает

Сформированный access токен (в виде строки)

См. определение в файле [JWT.cpp](#) строка 10

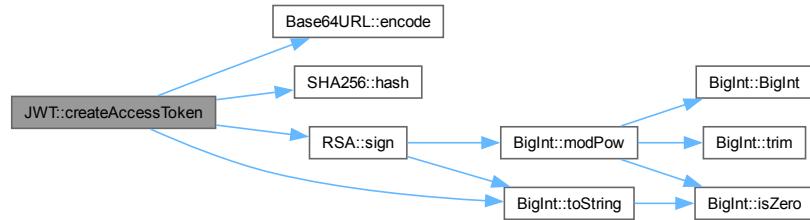
```
00010
00011     std::string headerStr = R"({ "alg": "RS256", "typ": "JWT" })";
00012     std::string headerEncoded = Base64URL::encode(headerStr);
00013
00014     uint64_t now = std::time(nullptr);
00015     uint64_t exp = now + expirationSeconds;
00016
00017     std::ostringstream payloadStream;
00018     payloadStream << "{\"sub\": \"" << subject << "\", \"iat\": " << now
00019             << "\", \"exp\": " << exp << "\", \"typ\": \"access\" }";
00020
00021     std::string payloadEncoded = Base64URL::encode(payloadStream.str());
00022
00023     std::string message = headerEncoded + "." + payloadEncoded;
00024     std::string hash = SHA256::hash(message);
00025
00026     BigInt hashInt(hash, 16);
00027     BigInt signatureInt = RSA::sign(hashInt, privKey);
00028     std::string signatureStr = Base64URL::encode(signatureInt.toString(16));
00029
00030     std::string token = message + "." + signatureStr;
00031
00032     std::cout << "[JWT::createAccessToken] ---" << std::endl;
00033     std::cout << "Header JSON: " << headerStr << std::endl;
00034     std::cout << "Payload JSON: " << payloadStream.str() << std::endl;
00035     std::cout << "Header Encoded: " << headerEncoded << std::endl;
00036     std::cout << "Payload Encoded: " << payloadEncoded << std::endl;
```

```

00037     std::cout << "Message (header.payload): " << message << std::endl;
00038     std::cout << "SHA256 Hash: " << hash << std::endl;
00039     std::cout << "Signature (hex): " << signatureInt.toString(16) << std::endl;
00040     std::cout << "Access Token: " << token << std::endl;
00041
00042     return token;
00043 }

```

Граф вызовов:



Граф вызова функции:



6.17.2.2 createRefreshToken()

```

std::string JWT::createRefreshToken (
    const std::string & subject,
    uint64_t expirationSeconds,
    const RSA privateKey & privKey) [static]

```

Создаёт refresh-токен для указанного пользователя.

Алгоритм аналогичен createAccessToken, но:

- В payload указывается тип токена "refresh".
- Время жизни устанавливается длиннее, чем у access-токена.

Аргументы

subject	Имя пользователя
expirationSeconds	Время жизни refresh токена (в секундах)
privKey	Приватный RSA-ключ

Возвращает

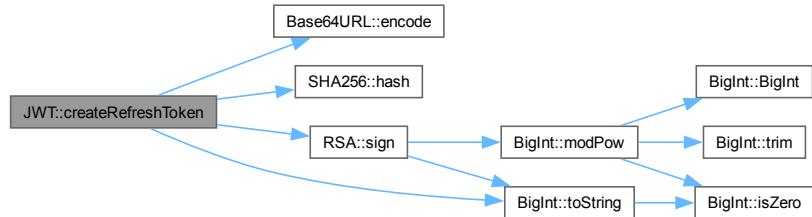
Сформированный refresh токен

См. определение в файле [JWT.cpp](#) строка 45

```

00045
00046     std::string headerStr = R"({"alg":"RS256","typ":"JWT"})";
00047     std::string headerEncoded = Base64URL::encode(headerStr);
00048
00049     uint64_t now = std::time(nullptr);
00050     uint64_t exp = now + expirationSeconds;
00051
00052     std::ostringstream payloadStream;
00053     payloadStream << "{\"sub\":\"" << subject << "\",\"iat\":"
00054         << now << ",\"exp\":"
00055         << exp << ",\"typ\":\"refresh\"}";
00056
00057     std::string payloadEncoded = Base64URL::encode(payloadStream.str());
00058
00059     std::string message = headerEncoded + "." + payloadEncoded;
00060     std::string hash = SHA256::hash(message);
00061
00062     BigInt hashInt(hash, 16);
00063     BigInt signatureInt = RSA::sign(hashInt, privKey);
00064     std::string signatureStr = Base64URL::encode(signatureInt.toString(16));
00065
00066     std::string token = message + "." + signatureStr;
00067
00068     std::cout << "[JWT::createRefreshToken] ---" << std::endl;
00069     std::cout << "Header JSON: " << headerStr << std::endl;
00070     std::cout << "Payload JSON: " << payloadEncoded << std::endl;
00071     std::cout << "Header Encoded: " << headerEncoded << std::endl;
00072     std::cout << "Payload Encoded: " << payloadEncoded << std::endl;
00073     std::cout << "Message (header.payload): " << message << std::endl;
00074     std::cout << "SHA256 Hash: " << hash << std::endl;
00075     std::cout << "Signature (hex): " << signatureStr << std::endl;
00076     std::cout << "Refresh Token: " << token << std::endl;
00077
00078     return token;
00079 }
```

Граф вызовов:



Граф вызова функции:



6.17.2.3 verifyAccessToken()

```
bool JWT::verifyAccessToken (
    const std::string & token,
    const RSA PublicKey & pubKey,
    std::string & outSubject) [static]
```

Проверяет access-токен на подлинность и срок действия.

Алгоритм:

1. Токен разбивается на три части: header, payload, signature.
2. Снова вычисляется SHA256-хеш от header.payload.
3. Проверяется RSA-подпись с использованием публичного ключа.
4. Проверяется тип токена ("typ": "access") в payload.
5. Проверяется время истечения.
6. Если всё верно, возвращается имя пользователя (sub).

Аргументы

token	JWT access-токен
pubKey	Публичный ключ для проверки подписи
outSubject	Выходной параметр — имя пользователя из токена

Возвращает

true, если токен валиден; false — иначе

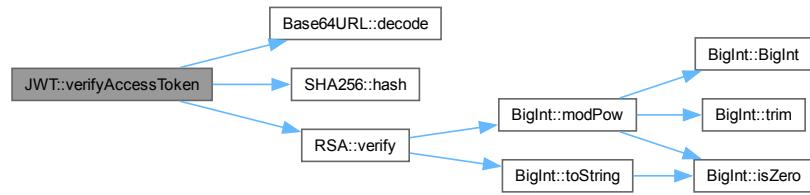
См. определение в файле [JWT.cpp](#) строка 80

```
00080
00081     std::cout << "[JWT::verifyAccessToken] ---" << std::endl;
00082     std::cout << "Received token: " << token << std::endl;
00083
00084     size_t firstDot = token.find('.');
00085     size_t secondDot = token.find('.', firstDot + 1);
00086     if (firstDot == std::string::npos || secondDot == std::string::npos) return false;
00087
00088     std::string headerB64 = token.substr(0, firstDot);
00089     std::string payloadB64 = token.substr(firstDot + 1, secondDot - firstDot - 1);
00090     std::string signatureB64 = token.substr(secondDot + 1);
00091
00092     std::string message = headerB64 + "." + payloadB64;
00093     std::string expectedHash = SHA256::hash(message);
00094
00095     std::string sigHex = Base64URL::decode(signatureB64);
00096     BigInt signature(sigHex, 16);
00097
00098     if (!RSA::verify(expectedHash, signature, pubKey)) {
00099         std::cerr << "[JWT] Подпись access токена недействительна" << std::endl;
00100         return false;
00101     }
00102
00103     std::string payloadJson = Base64URL::decode(payloadB64);
00104     std::cout << "Decoded payload: " << payloadJson << std::endl;
00105
00106     if (payloadJson.find("\\" typ \":\" access \") == std::string::npos) {
00107         std::cerr << "[JWT] Токен не является access" << std::endl;
00108         return false;
00109     }
00110
00111     size_t subPos = payloadJson.find("\\" sub \":\"");
00112     size_t expPos = payloadJson.find("\\" exp \":");
```

```

00113
00114     if (subPos == std::string::npos || expPos == std::string::npos) return false;
00115
00116     subPos += 7;
00117     size_t subEnd = payloadJson.find("\\\"", subPos);
00118     outSubject = payloadJson.substr(subPos, subEnd - subPos);
00119
00120     expPos += 6;
00121     size_t expEnd = payloadJson.find_first_of("},{", expPos);
00122     std::string expStr = payloadJson.substr(expPos, expEnd - expPos);
00123     uint64_t exp = std::stoull(expStr);
00124
00125     std::cout << "Subject: " << outSubject << std::endl;
00126     std::cout << "Expiration: " << exp << ", now: " << std::time(nullptr) << std::endl;
00127
00128     if (static_cast<uint64_t>(std::time(nullptr)) > exp) {
00129         std::cerr << "[JWT] Access токен просрочен" << std::endl;
00130         return false;
00131     }
00132
00133     return true;
00134 }
```

Граф вызовов:



Граф вызова функции:



6.17.2.4 verifyRefreshToken()

```

bool JWT::verifyRefreshToken (
    const std::string & token,
    const RSA PublicKey & pubKey,
    std::string & outSubject) [static]
```

Проверяет refresh-токен на подлинность и срок действия.

Алгоритм аналогичен verifyAccessToken, но с проверкой "typ": "refresh".

Аргументы

token	JWT refresh-токен
-------	-------------------

pubKey	Публичный RSA-ключ
outSubject	Выходной параметр — имя пользователя

Возвращает

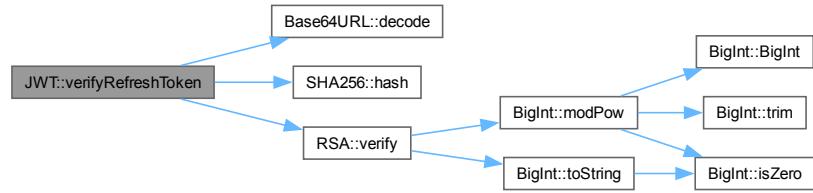
true, если токен валиден и не просрочен; false — иначе

См. определение в файле [JWT.cpp](#) строка 136

```

00136
00137     std::cout << "[JWT::verifyRefreshToken] ---" << std::endl;
00138     std::cout << "Received token: " << token << std::endl;
00139
00140     size_t firstDot = token.find('.');
00141     size_t secondDot = token.find('.', firstDot + 1);
00142     if (firstDot == std::string::npos || secondDot == std::string::npos) return false;
00143
00144     std::string headerB64 = token.substr(0, firstDot);
00145     std::string payloadB64 = token.substr(firstDot + 1, secondDot - firstDot - 1);
00146     std::string signatureB64 = token.substr(secondDot + 1);
00147
00148     std::string message = headerB64 + "." + payloadB64;
00149     std::string expectedHash = SHA256::hash(message);
00150
00151     std::string sigHex = Base64URL::decode(signatureB64);
00152     BigInt signature(sigHex, 16);
00153
00154     if (!RSA::verify(expectedHash, signature, pubKey)) {
00155         std::cerr << "[JWT] Refresh подпись недействительна" << std::endl;
00156         return false;
00157     }
00158
00159     std::string payloadJson = Base64URL::decode(payloadB64);
00160     std::cout << "Decoded payload: " << payloadJson << std::endl;
00161
00162     if (payloadJson.find("\\"typ\":\\refresh\\") == std::string::npos) {
00163         std::cerr << "[JWT] Токен не является refresh" << std::endl;
00164         return false;
00165     }
00166
00167     size_t subPos = payloadJson.find("\\\"sub\\\":\\\"");
00168     size_t expPos = payloadJson.find("\\\"exp\\\":");
00169     if (subPos == std::string::npos || expPos == std::string::npos) return false;
00170
00171     subPos += 7;
00172     size_t subEnd = payloadJson.find("\\\"", subPos);
00173     outSubject = payloadJson.substr(subPos, subEnd - subPos);
00174
00175     expPos += 6;
00176     size_t expEnd = payloadJson.find_first_of(",}", expPos);
00177     std::string expStr = payloadJson.substr(expPos, expEnd - expPos);
00178     uint64_t exp = std::stoull(expStr);
00179
00180     std::cout << "Subject: " << outSubject << std::endl;
00181     std::cout << "Expiration: " << exp << ", now: " << std::time(nullptr) << std::endl;
00182
00183     if (static_cast<uint64_t>(std::time(nullptr)) > exp) {
00184         std::cerr << "[JWT] Refresh токен просрочен" << std::endl;
00185         return false;
00186     }
00187
00188     return true;
00189 }
```

Граф вызовов:



Граф вызова функции:



Объявления и описания членов классов находятся в файлах:

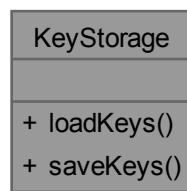
- [JWT.h](#)
- [JWT.cpp](#)

6.18 Класс KeyStorage

Класс для хранения и загрузки RSA-ключей на диск.

```
#include <KeyStorage.h>
```

Граф связей класса KeyStorage:



Открытые статические члены

- static bool `loadKeys` (`RSAPublicKey` &`pubKey`, `RSAPrivateKey` &`privKey`)
Загружает RSA-ключи из файлов `rsa_private.key` и `rsa_public.key`.
- static void `saveKeys` (const `RSAPublicKey` &`pubKey`, const `RSAPrivateKey` &`privKey`)
Сохраняет RSA-ключи в файлы.

6.18.1 Подробное описание

Класс для хранения и загрузки RSA-ключей на диск.

Данный класс предоставляет интерфейс для:

- сохранения приватного и публичного ключа в текстовые файлы (`rsa_private.key` и `rsa_public.key`);
- верификации целостности приватного ключа с помощью SHA-256 хеша;
- загрузки ключей с диска и проверки корректности формата.

Формат файлов:

- Приватный ключ (`rsa_private.key`) содержит строку вида:

```
d;n  
hash=SHA256(d;n)
```

Это позволяет проверить, что приватный ключ не был случайно или намеренно изменён.

- Публичный ключ (`rsa_public.key`) содержит строку вида:

```
e;n
```

См. определение в файле `KeyStorage.h` строка [25](#)

6.18.2 Методы

6.18.2.1 `loadKeys()`

```
bool KeyStorage::loadKeys (
    RSAPublicKey & pubKey,
    RSAPrivatekey & privKey) [static]
```

Загружает RSA-ключи из файлов `rsa_private.key` и `rsa_public.key`.

Аргументы

out	<code>pubKey</code>	Публичный ключ (заполняется из файла)
out	<code>privKey</code>	Приватный ключ (заполняется из файла)

Возвращает

true, если ключи успешно загружены и верифицированы; false в случае ошибки или несоответствия контрольной суммы.

См. определение в файле [KeyStorage.cpp](#) строка 25

```

00025      {
00026      std::ifstream privIn(PRIV_FILE);
00027      std::ifstream pubIn(PUB_FILE);
00028
00029      if (!privIn || !pubIn) return false;
00030
00031      std::string privLine, hashLine;
00032      if (!std::getline(privIn, privLine) || !std::getline(privIn, hashLine)) return false;
00033
00034      // validate hash
00035      std::string expectedHash = SHA256::hash(privLine);
00036      if (hashLine != "hash=" + expectedHash) {
00037          std::cerr << "[KeyStorage] Ошибка: контрольная сумма не совпадает. Приватный ключ поврежден." << std::endl;
00038          return false;
00039      }
00040
00041      size_t delimPos = privLine.find(';');
00042      if (delimPos == std::string::npos) return false;
00043      std::string dStr = privLine.substr(0, delimPos);
00044      std::string nStr = privLine.substr(delimPos + 1);
00045
00046      privKey.d = BigInt(dStr);
00047      privKey.n = BigInt(nStr);
00048
00049      // Load public key
00050      std::string pubLine;
00051      if (!std::getline(pubIn, pubLine)) return false;
00052
00053      delimPos = pubLine.find(';');
00054      if (delimPos == std::string::npos) return false;
00055      std::string eStr = pubLine.substr(0, delimPos);
00056      std::string pubNStr = pubLine.substr(delimPos + 1);
00057
00058      pubKey.e = BigInt(eStr);
00059      pubKey.n = BigInt(pubNStr);
00060
00061      return true;
00062 }
```

Граф вызовов:



Граф вызова функции:



6.18.2.2 saveKeys()

```
void KeyStorage::saveKeys (
    const RSA PublicKey & publicKey,
    const RSA PrivateKey & privateKey) [static]
```

Сохраняет RSA-ключи в файлы.

Приватный ключ сохраняется с добавлением SHA-256 контрольной суммы, чтобы при последующей загрузке можно было проверить его целостность.

Аргументы

pubKey	Публичный ключ, который сохраняется в rsa_public.key
privKey	Приватный ключ, который сохраняется в rsa_private.key с хешем

См. определение в файле [KeyStorage.cpp](#) строка 9

```
00009                                     {
00010     // Save private key with SHA256 hash
00011     std::ofstream privOut(PRIV_FILE);
00012     if (privOut) {
00013         std::string content = privKey.d.toString() + ";" + privKey.n.toString();
00014         std::string hash = SHA256::hash(content);
00015         privOut << content << "\n" << "hash=" << hash << "\n";
00016     }
00017
00018     // Save public key
00019     std::ofstream pubOut(PUB_FILE);
00020     if (pubOut) {
00021         pubOut << publicKey.e.toString() << ";" << publicKey.n.toString() << "\n";
00022     }
00023 }
```

Граф вызовов:



Граф вызова функции:



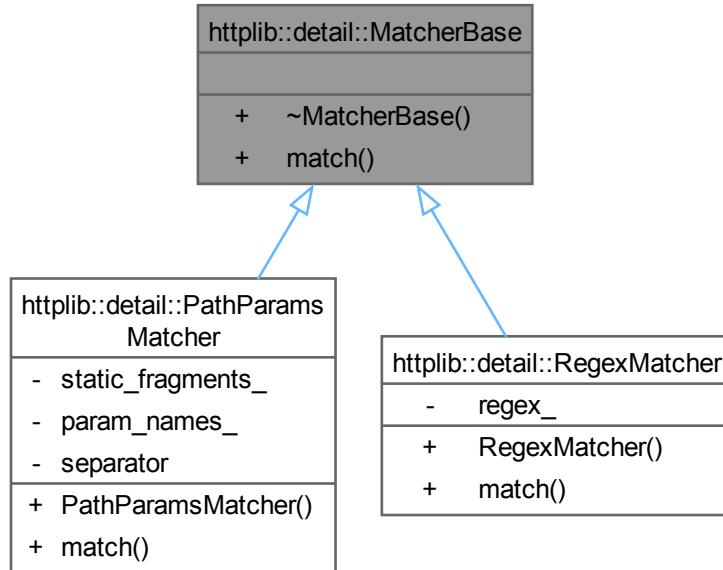
Объявления и описания членов классов находятся в файлах:

- [KeyStorage.h](#)
- [KeyStorage.cpp](#)

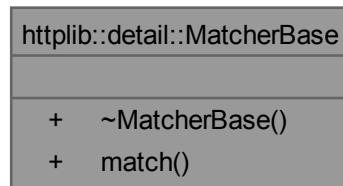
6.19 Класс httpplib::detail::MatcherBase

```
#include <httpplib.h>
```

Граф наследования: httpplib::detail::MatcherBase:



Граф связей класса httpplib::detail::MatcherBase:



Открытые члены

- virtual `~MatcherBase ()=default`
- virtual bool `match (Request &request) const =0`

6.19.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 879

6.19.2 Конструктор(ы)

6.19.2.1 `~MatcherBase()`

```
virtual httpplib::detail::MatcherBase::~MatcherBase () [virtual], [default]
```

6.19.3 Методы

6.19.3.1 `match()`

```
virtual bool httpplib::detail::MatcherBase::match (
    Request & request) const [pure virtual]
```

Замещается в [httpplib::detail::PathParamsMatcher](#) и [httpplib::detail::RegexMatcher](#).

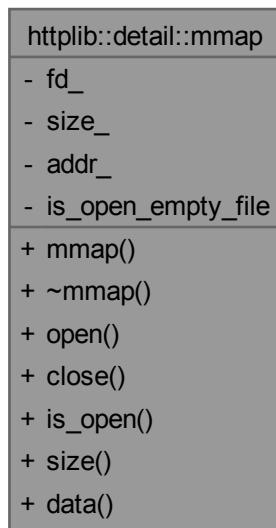
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.20 Класс `httpplib::detail::mmap`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::mmap`:



Открытые члены

- `mmap` (`const char *path`)
- `~mmap` ()
- `bool open` (`const char *path`)
- `void close` ()
- `bool is_open` () const
- `size_t size` () const
- `const char * data` () const

Закрытые данные

- `int fd_ = -1`
- `size_t size_ = 0`
- `void * addr_ = nullptr`
- `bool is_open_empty_file = false`

6.20.1 Подробное описание

См. определение в файле `httpplib.h` строка 2617

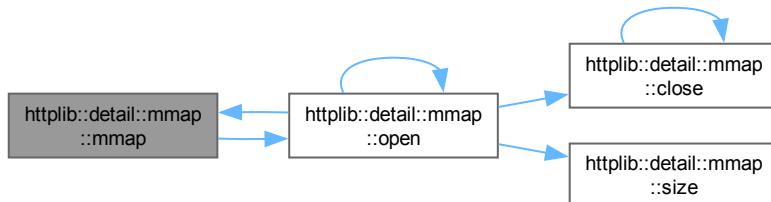
6.20.2 Конструктор(ы)

6.20.2.1 `mmap()`

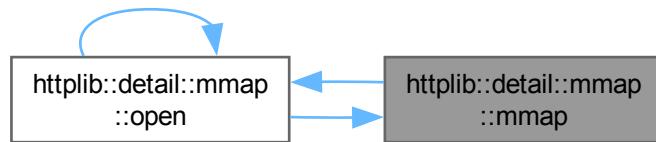
```
httpplib::detail::mmap::mmap (
    const char * path) [inline]
```

См. определение в файле `httpplib.h` строка 3109
03109 { `open(path);` }

Граф вызовов:



Граф вызова функции:



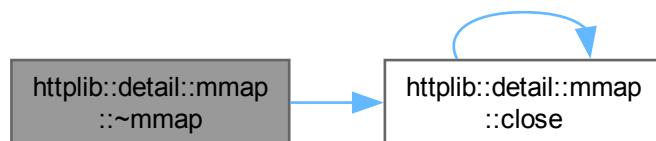
6.20.2.2 `~mmap()`

`httpplib::detail::mmap::~mmap ()` [inline]

См. определение в файле [httpplib.h](#) строка 3111

03111 { `close();` }

Граф вызовов:



6.20.3 Методы

6.20.3.1 `close()`

`void httpplib::detail::mmap::close ()` [inline]

См. определение в файле [httpplib.h](#) строка 3205

```

03205 {
03206 #if defined(_WIN32)
03207 if (addr_) {
03208     ::UnmapViewOfFile(addr_);
03209     addr_ = nullptr;
03210 }
03211 if (hMapping_) {
03212     ::CloseHandle(hMapping_);
03213     hMapping_ = NULL;
03214 }
03215 if (hFile != INVALID_HANDLE_VALUE) {
03216     ::CloseHandle(hFile_);
  
```

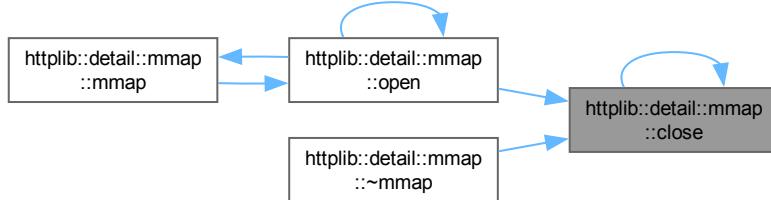
```

03219     hFile_ = INVALID_HANDLE_VALUE;
03220 }
03221
03222 if (is_open_empty_file == false)
03223 #else
03224 if (addr_ != nullptr) {
03225     munmap(addr_, size_);
03226     addr_ = nullptr;
03227 }
03228
03229 if (fd_ != -1) {
03230     ::close(fd_);
03231     fd_ = -1;
03232 }
03233 #endif
03234 size_ = 0;
03235 }
```

Граф вызовов:



Граф вызова функции:



6.20.3.2 `data()`

`const char * httpplib::detail::mmap::data () const [inline]`

См. определение в файле [httpplib.h](#) строка 3201

```

03201 {
03202     return is_open_empty_file ? "" : static_cast<const char *>(addr_);
03203 }
```

6.20.3.3 `is_open()`

`bool httpplib::detail::mmap::is_open () const [inline]`

См. определение в файле [httpplib.h](#) строка 3195

```

03195 {
03196     return is_open_empty_file ? true : addr_ != nullptr;
03197 }
```

6.20.3.4 `open()`

```
bool httpplib::detail::mmap::open (
    const char * path) [inline]
```

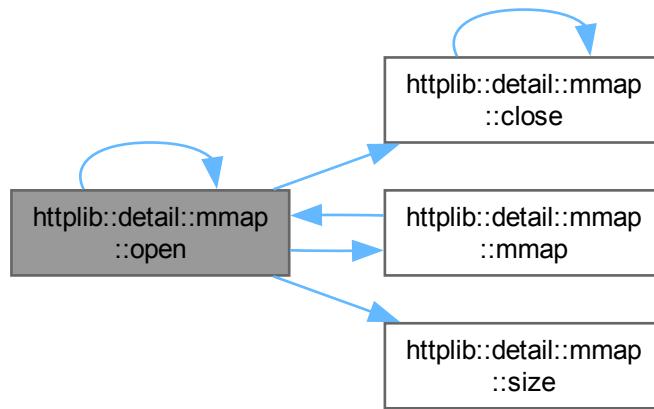
См. определение в файле [httpplib.h](#) строка 3113

```
03113     {
03114     close();
03115
03116 #if defined(_WIN32)
03117     auto wpath = u8string_to_wstring(path);
03118     if (wpath.empty()) { return false; }
03119
03120 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03121     hFile_ = ::CreateFile2(wpath.c_str(), GENERIC_READ, FILE_SHARE_READ,
03122                             OPEN_EXISTING, NULL);
03123 #else
03124     hFile_ = ::CreateFileW(wpath.c_str(), GENERIC_READ, FILE_SHARE_READ, NULL,
03125                             OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
03126 #endif
03127
03128     if (hFile_ == INVALID_HANDLE_VALUE) { return false; }
03129
03130     LARGE_INTEGER size{};
03131     if (!::GetFileSizeEx(hFile_, &size)) { return false; }
03132     // If the following line doesn't compile due to QuadPart, update Windows SDK.
03133     // See:
03134     // https://github.com/yhirose/cpp-httplib/issues/1903#issuecomment-2316520721
03135     if (static_cast<ULLONG>(size.QuadPart) >
03136         (std::numeric_limits<decltype(size)>::max)()) {
03137         // `size_t` might be 32-bits, on 32-bits Windows.
03138         return false;
03139     }
03140     size_ = static_cast<size_t>(size.QuadPart);
03141
03142 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03143     hMapping_ =
03144         ::CreateFileMappingFromApp(hFile_, NULL, PAGE_READONLY, size_, NULL);
03145 #else
03146     hMapping_ = ::CreateFileMappingW(hFile_, NULL, PAGE_READONLY, 0, 0, NULL);
03147 #endif
03148
03149     // Special treatment for an empty file...
03150     if (hMapping_ == NULL && size_ == 0) {
03151         close();
03152         is_open_empty_file = true;
03153         return true;
03154     }
03155
03156     if (hMapping_ == NULL) {
03157         close();
03158         return false;
03159     }
03160
03161 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03162     addr_ = ::MapViewOfFileFromApp(hMapping_, FILE_MAP_READ, 0, 0);
03163 #else
03164     addr_ = ::MapViewOfFile(hMapping_, FILE_MAP_READ, 0, 0, 0);
03165 #endif
03166
03167     if (addr_ == nullptr) {
03168         close();
03169         return false;
03170     }
03171 #else
03172     fd_ = ::open(path, O_RDONLY);
03173     if (fd_ == -1) { return false; }
03174
03175     struct stat sb;
03176     if (fstat(fd_, &sb) == -1) {
03177         close();
03178         return false;
03179     }
03180     size_ = static_cast<size_t>(sb.st_size);
03181
03182     addr_ = ::mmap(NULL, size_, PROT_READ, MAP_PRIVATE, fd_, 0);
03183
03184     // Special treatment for an empty file...
03185     if (addr_ == MAP_FAILED && size_ == 0) {
03186         close();
03187         is_open_empty_file = true;
03188         return false;
03189     }
03190 #endif
```

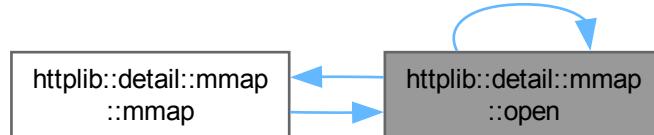
```

03191
03192   return true;
03193 }
```

Граф вызовов:



Граф вызова функции:



6.20.3.5 `size()`

`size_t httpplib::detail::mmap::size () const [inline]`

См. определение в файле [httpplib.h](#) строка 3199

```
03199 { return size_; }
```

Граф вызова функции:



6.20.4 Данные класса

6.20.4.1 `addr_`

`void* httpplib::detail::mmap::addr_ = nullptr [private]`

См. определение в файле [httpplib.h](#) строка [2637](#)

6.20.4.2 `fd_`

`int httpplib::detail::mmap::fd_ = -1 [private]`

См. определение в файле [httpplib.h](#) строка [2634](#)

6.20.4.3 `is_open_empty_file`

`bool httpplib::detail::mmap::is_open_empty_file = false [private]`

См. определение в файле [httpplib.h](#) строка [2638](#)

6.20.4.4 `size_`

`size_t httpplib::detail::mmap::size_ = 0 [private]`

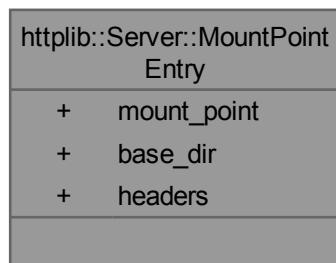
См. определение в файле [httpplib.h](#) строка [2636](#)

Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.21 Структура `httpplib::Server::MountPointEntry`

Граф связей класса `httpplib::Server::MountPointEntry`:



Открытые атрибуты

- `std::string mount_point`
- `std::string base_dir`
- `Headers headers`

6.21.1 Подробное описание

См. определение в файле [httpplib.h](#) строка [1123](#)

6.21.2 Данные класса

6.21.2.1 `base_dir`

`std::string httpplib::Server::MountPointEntry::base_dir`

См. определение в файле [httpplib.h](#) строка [1125](#)

6.21.2.2 `headers`

`Headers httpplib::Server::MountPointEntry::headers`

См. определение в файле [httpplib.h](#) строка [1126](#)

6.21.2.3 `mount_point`

`std::string httpplib::Server::MountPointEntry::mount_point`

См. определение в файле [httpplib.h](#) строка [1124](#)

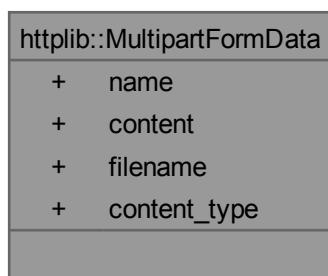
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.22 Структура `httpplib::MultipartFormData`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::MultipartFormData`:



Открытые атрибуты

- `std::string name`
- `std::string content`
- `std::string filename`
- `std::string content_type`

6.22.1 Подробное описание

См. определение в файле [httpplib.h](#) строка [533](#)

6.22.2 Данные класса

6.22.2.1 `content`

`std::string httpplib::MultipartFormData::content`

См. определение в файле [httpplib.h](#) строка [535](#)

6.22.2.2 `content_type`

`std::string httpplib::MultipartFormData::content_type`

См. определение в файле [httpplib.h](#) строка [537](#)

6.22.2.3 `filename`

`std::string httpplib::MultipartFormData::filename`

См. определение в файле [httpplib.h](#) строка [536](#)

6.22.2.4 `name`

`std::string httpplib::MultipartFormData::name`

См. определение в файле [httpplib.h](#) строка [534](#)

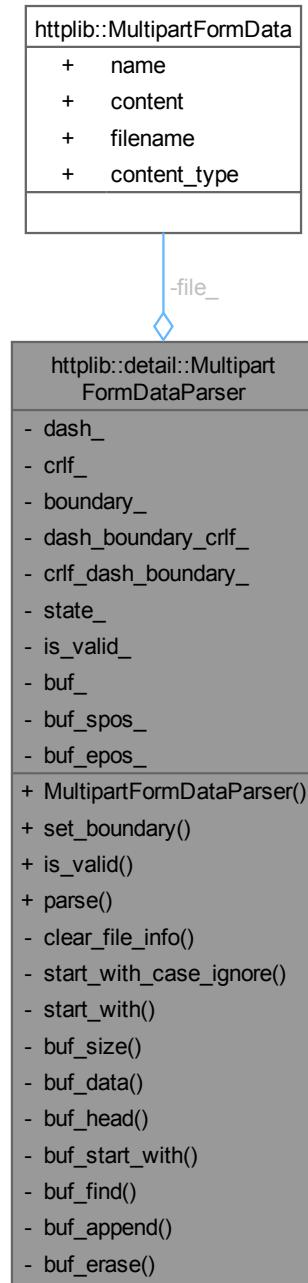
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.23 Класс `httpplib::detail::MultipartFormDataParser`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::MultipartFormDataParser`:



Открытые члены

- `MultipartFormDataParser ()=default`

- void `set_boundary` (std::string &&boundary)
- bool `is_valid` () const
- bool `parse` (const char *buf, size_t n, const ContentReceiver &content_callback, const MultipartContentHeader &header_callback)

Закрытые члены

- void `clear_file_info` ()
- bool `start_with_case_ignore` (const std::string &a, const char *b) const
- bool `start_with` (const std::string &a, size_t spos, size_t epos, const std::string &b) const
- size_t `buf_size` () const
- const char * `buf_data` () const
- std::string `buf_head` (size_t l) const
- bool `buf_start_with` (const std::string &s) const
- size_t `buf_find` (const std::string &s) const
- void `buf_append` (const char *data, size_t n)
- void `buf_erase` (size_t size)

Закрытые данные

- const std::string `dash_` = "--"
- const std::string `crlf_` = "\r\n"
- std::string `boundary_`
- std::string `dash_boundary_crlf_`
- std::string `crlf_dash_boundary_`
- size_t `state_` = 0
- bool `is_valid_` = false
- MultipartFormData `file_`
- std::string `buf_`
- size_t `buf_spos_` = 0
- size_t `buf_epos_` = 0

6.23.1 Подробное описание

См. определение в файле `httpplib.h` строка [4964](#)

6.23.2 Конструктор(ы)

6.23.2.1 `MultipartFormDataParser()`

`httpplib::detail::MultipartFormDataParser::MultipartFormDataParser () [default]`

6.23.3 Методы

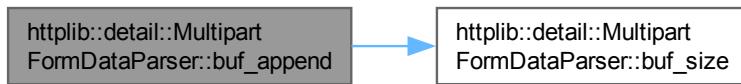
6.23.3.1 `buf_append()`

```
void httpplib::detail::MultipartFormDataParser::buf_append (
    const char * data,
    size_t n) [inline], [private]
```

См. определение в файле [httpplib.h](#) строка 5184

```
05184     {
05185     auto remaining_size = buf_size();
05186     if (remaining_size > 0 && buf_spos_ > 0) {
05187         for (size_t i = 0; i < remaining_size; i++) {
05188             buf_[i] = buf_[buf_spos_ + i];
05189         }
05190     }
05191     buf_spos_ = 0;
05192     buf_epos_ = remaining_size;
05193
05194     if (remaining_size + n > buf_.size()) { buf_.resize(remaining_size + n); }
05195
05196     for (size_t i = 0; i < n; i++) {
05197         buf_[buf_epos_ + i] = data[i];
05198     }
05199     buf_epos_ += n;
05200 }
```

Граф вызовов:



Граф вызова функции:



6.23.3.2 `buf_data()`

```
const char * httpplib::detail::MultipartFormDataParser::buf_data () const [inline], [private]
```

См. определение в файле [httpplib.h](#) строка 5153

```
05153 { return &buf_[buf_spos_]; }
```

Граф вызова функции:



6.23.3.3 `buf_erase()`

```
void httpplib::detail::MultipartFormDataParser::buf_erase (
    size_t size) [inline], [private]
```

См. определение в файле [httpplib.h](#) строка [5202](#)

```
05202 { buf_spos_ += size; }
```

Граф вызова функции:



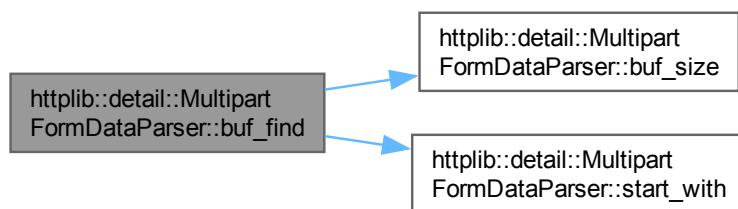
6.23.3.4 `buf_find()`

```
size_t httpplib::detail::MultipartFormDataParser::buf_find (
    const std::string & s) const [inline], [private]
```

См. определение в файле [httpplib.h](#) строка [5161](#)

```
05161
05162     auto c = s.front();
05163
05164     size_t off = buf_spos_;
05165     while (off < buf_epos_) {
05166         auto pos = off;
05167         while (true) {
05168             if (pos == buf_epos_) { return buf_size(); }
05169             if (buf_[pos] == c) { break; }
05170             pos++;
05171         }
05172
05173         auto remaining_size = buf_epos_ - pos;
05174         if (s.size() > remaining_size) { return buf_size(); }
05175
05176         if (start_with(buf_, pos, buf_epos_, s)) { return pos - buf_spos_; }
05177
05178         off = pos + 1;
05179     }
05180
05181     return buf_size();
05182 }
```

Граф вызовов:



Граф вызова функции:



6.23.3.5 buf_head()

```
std::string httpplib::detail::MultipartFormDataParser::buf_head (
    size_t l) const [inline], [private]
```

См. определение в файле `httpplib.h` строка 5155
05155 { `return buf_.substr(buf_spos_, l);` }

Граф вызова функции:

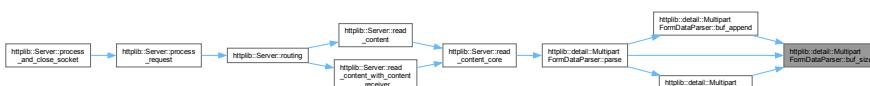


6.23.3.6 buf_size()

```
size_t httpplib::detail::MultipartFormDataParser::buf_size () const [inline], [private]
```

См. определение в файле `httpplib.h` строка 5151
05151 { `return buf_epos_ - buf_spos_;` }

Граф вызова функции:



6.23.3.7 buf_start_with()

```
bool httpplib::detail::MultipartFormDataParser::buf_start_with (
    const std::string & s) const [inline], [private]
```

См. определение в файле `httpplib.h` строка 5157
05157 {
05158 `return start_with(buf_, buf_spos_, buf_epos_, s);`
05159 }

Граф вызовов:



Граф вызова функции:



6.23.3.8 clear_file_info()

`void httpplib::detail::MultipartFormDataParser::clear_file_info () [inline], [private]`

См. определение в файле [httpplib.h](#) строка 5114

```

05114     {
05115     file_.name.clear();
05116     file_.filename.clear();
05117     file_.content_type.clear();
05118 }
```

Граф вызова функции:



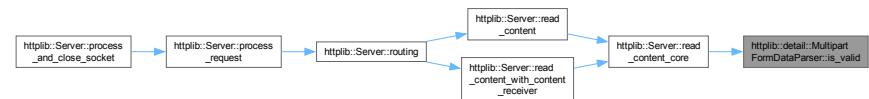
6.23.3.9 is_valid()

`bool httpplib::detail::MultipartFormDataParser::is_valid () const [inline]`

См. определение в файле [httpplib.h](#) строка 4974

```
04974 { return is_valid_; }
```

Граф вызова функции:



6.23.3.10 `parse()`

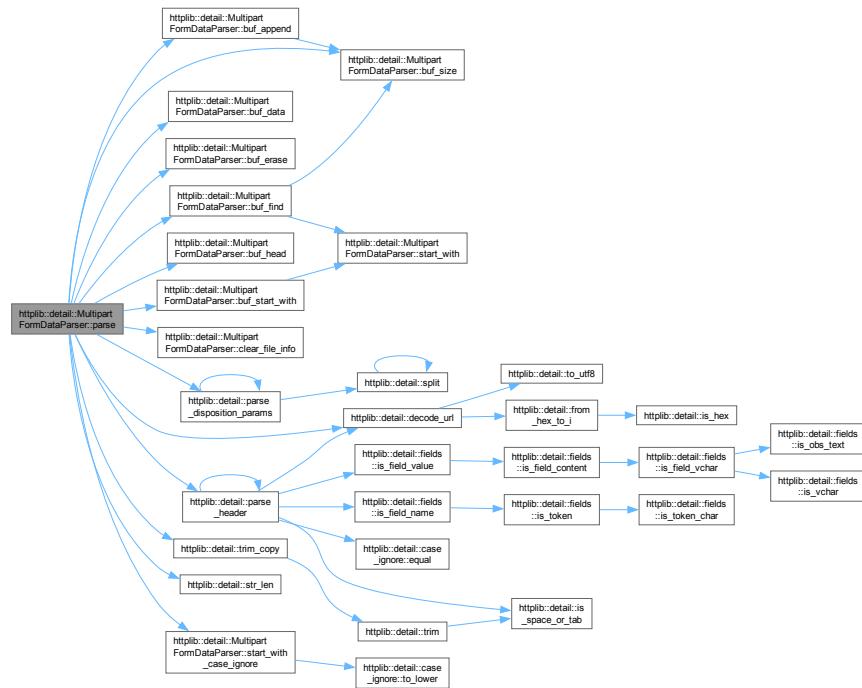
```
bool httpplib::detail::MultipartFormDataParser::parse (
    const char * buf,
    size_t n,
    const ContentReceiver & content_callback,
    const MultipartContentHeader & header_callback) [inline]
```

См. определение в файле `httpplib.h` строка 4976

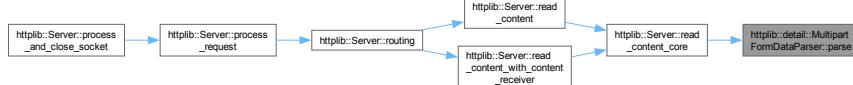
```
04977
04978
04979     buf_append(buf, n);
04980
04981     while (buf_size() > 0) {
04982         switch (state_) {
04983             case 0: { // Initial boundary
04984                 buf_erase(buf_find(dash_boundary_crlf_));
04985                 if (!dash_boundary_crlf_.size() > buf_size()) { return true; }
04986                 if (!buf_start_with(dash_boundary_crlf_)) { return false; }
04987                 buf_erase(dash_boundary_crlf_.size());
04988                 state_ = 1;
04989                 break;
04990             }
04991             case 1: { // New entry
04992                 clear_file_info();
04993                 state_ = 2;
04994                 break;
04995             }
04996             case 2: { // Headers
04997                 auto pos = buf_find(crlf_);
04998                 if (pos > CPPHTTPLIB_HEADER_MAX_LENGTH) { return false; }
04999                 while (pos < buf_size()) {
05000                     // Empty line
05001                     if (pos == 0) {
05002                         if (!header_callback(file_)) {
05003                             is_valid_ = false;
05004                             return false;
05005                         }
05006                         buf_erase(crlf_.size());
05007                         state_ = 3;
05008                         break;
05009                     }
05010
05011                     const auto header = buf_head(pos);
05012
05013                     if (!parse_header(header.data(), header.data() + header.size(),
05014                                     [&](const std::string &, const std::string &) {})) {
05015                         is_valid_ = false;
05016                         return false;
05017                     }
05018
05019                     constexpr const char header_content_type[] = "Content-Type:";
05020
05021                     if (start_with_case_ignore(header, header_content_type)) {
05022                         file_.content_type =
05023                             trim_copy(header.substr(str_len(header_content_type)));
05024                     } else {
05025                         thread_local const std::regex re_content_disposition(
05026                             R"~(Content-Disposition:\s*form-data;\s*(.*))~",
05027                             std::regex_constants::icase);
05028
05029                         std::smatch m;
05030                         if (std::regex_match(header, m, re_content_disposition)) {
05031                             Params params;
05032                             parse_disposition_params(m[1], params);
05033
05034                             auto it = params.find("name");
05035                             if (it != params.end()) {
05036                                 file_.name = it->second;
05037                             } else {
05038                                 is_valid_ = false;
05039                                 return false;
05040                             }
05041
05042                             it = params.find("filename");
05043                             if (it != params.end()) { file_.filename = it->second; }
05044
05045                             it = params.find("filename*");
05046                             if (it != params.end()) {
05047                                 // Only allow UTF-8 encoding...
05048                                 thread_local const std::regex re_rfc5987_encoding(
05049                                     R"~(UTF-8"(.+?))~", std::regex_constants::icase);
05050                             }
05051                         }
05052                     }
05053                 }
05054             }
05055         }
05056     }
05057 }
```

```
05050         std::smatch m2;
05051     if (std::regex_match(it->second, m2, re_rfc5987_encoding)) {
05052         file_.filename = decode_url(m2[1], false); // override...
05053     } else {
05054         is_valid_ = false;
05055         return false;
05056     }
05057 }
05058 }
05059 }
05060 buf_.erase(pos + crlf_.size());
05061 pos = buf_.find(crlf_);
05062 }
05063 if (state_ != 3) { return true; }
05064 break;
05065 }
05066 case 3: { // Body
05067     if (crlf_dash_boundary_.size() > buf_size()) { return true; }
05068     auto pos = buf_.find(crlf_dash_boundary_);
05069     if (pos < buf_size()) {
05070         if (!content_callback(buf_data(), pos)) {
05071             is_valid_ = false;
05072             return false;
05073         }
05074         buf_.erase(pos + crlf_dash_boundary_.size());
05075         state_ = 4;
05076     } else {
05077         auto len = buf_size() - crlf_dash_boundary_.size();
05078         if (len > 0) {
05079             if (!content_callback(buf_data(), len)) {
05080                 is_valid_ = false;
05081                 return false;
05082             }
05083             buf_.erase(len);
05084         }
05085         return true;
05086     }
05087 }
05088 break;
05089 }
05090 case 4: { // Boundary
05091     if (crlf_.size() > buf_size()) { return true; }
05092     if (buf_start_with(crlf_)) {
05093         buf_.erase(crlf_.size());
05094         state_ = 1;
05095     } else {
05096         if (dash_.size() > buf_size()) { return true; }
05097         if (buf_start_with(dash_)) {
05098             buf_.erase(dash_.size());
05099             is_valid_ = true;
05100             buf_.erase(buf_size()); // Remove epilogue
05101         } else {
05102             return true;
05103         }
05104     }
05105     break;
05106 }
05107 }
05108 }
05109 return true;
05110 }
```

Граф вызовов:



Граф вызова функции:



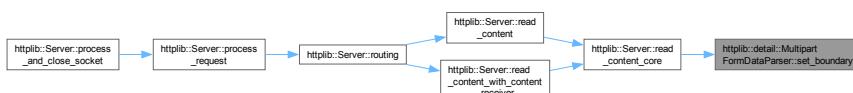
6.23.3.11 set_boundary()

```
void httpplib::detail::MultipartFormDataParser::set_boundary (
    std::string && boundary) [inline]
```

См. определение в файле `httpplib.h` строка 4968

```
04968     {
04969     boundary_ = boundary;
04970     dash_boundary_crlf_ = dash_ + boundary_ + crlf_;
04971     crlf_dash_boundary_ = crlf_ + dash_ + boundary_;
04972 }
```

Граф вызова функции:



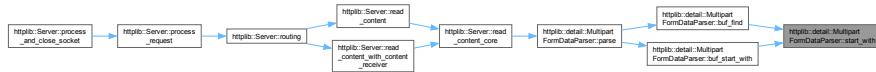
6.23.3.12 `start_with()`

```
bool httpplib::detail::MultipartFormDataParser::start_with (
    const std::string & a,
    size_t spos,
    size_t epos,
    const std::string & b) const [inline], [private]
```

См. определение в файле `httpplib.h` строка 5142

```
05143     {
05144     if (epos - spos < b.size()) { return false; }
05145     for (size_t i = 0; i < b.size(); i++) {
05146         if (a[i + spos] != b[i]) { return false; }
05147     }
05148     return true;
05149 }
```

Граф вызова функции:



6.23.3.13 `start_with_case_ignore()`

```
bool httpplib::detail::MultipartFormDataParser::start_with_case_ignore (
    const std::string & a,
    const char * b) const [inline], [private]
```

См. определение в файле `httpplib.h` строка 5120

```
05120 {
05121     const auto b_len = strlen(b);
05122     if (a.size() < b_len) { return false; }
05123     for (size_t i = 0; i < b_len; i++) {
05124         if (case_ignore::to_lower(a[i]) != case_ignore::to_lower(b[i])) {
05125             return false;
05126         }
05127     }
05128     return true;
05129 }
```

Граф вызовов:



Граф вызова функции:



6.23.4 Данные класса

6.23.4.1 `boundary_`

```
std::string httpplib::detail::MultipartFormDataParser::boundary_ [private]
```

См. определение в файле [httpplib.h](#) строка [5133](#)

6.23.4.2 `buf_`

```
std::string httpplib::detail::MultipartFormDataParser::buf_ [private]
```

См. определение в файле [httpplib.h](#) строка [5204](#)

6.23.4.3 `buf_epos_`

```
size_t httpplib::detail::MultipartFormDataParser::buf_epos_ = 0 [private]
```

См. определение в файле [httpplib.h](#) строка [5206](#)

6.23.4.4 `buf_spos_`

```
size_t httpplib::detail::MultipartFormDataParser::buf_spos_ = 0 [private]
```

См. определение в файле [httpplib.h](#) строка [5205](#)

6.23.4.5 `crlf_`

```
const std::string httpplib::detail::MultipartFormDataParser::crlf_ = "\r\n" [private]
```

См. определение в файле [httpplib.h](#) строка [5132](#)

6.23.4.6 `crlf_dash_boundary_`

```
std::string httpplib::detail::MultipartFormDataParser::crlf_dash_boundary_ [private]
```

См. определение в файле [httpplib.h](#) строка [5135](#)

6.23.4.7 `dash_`

```
const std::string httpplib::detail::MultipartFormDataParser::dash_ = "--" [private]
```

См. определение в файле [httpplib.h](#) строка [5131](#)

6.23.4.8 `dash_boundary_crlf_`

`std::string httpplib::detail::MultipartFormDataParser::dash_boundary_crlf_ [private]`

См. определение в файле [httpplib.h](#) строка [5134](#)

6.23.4.9 `file_`

`MultipartFormData httpplib::detail::MultipartFormDataParser::file_ [private]`

См. определение в файле [httpplib.h](#) строка [5139](#)

6.23.4.10 `is_valid_`

`bool httpplib::detail::MultipartFormDataParser::is_valid_ = false [private]`

См. определение в файле [httpplib.h](#) строка [5138](#)

6.23.4.11 `state_`

`size_t httpplib::detail::MultipartFormDataParser::state_ = 0 [private]`

См. определение в файле [httpplib.h](#) строка [5137](#)

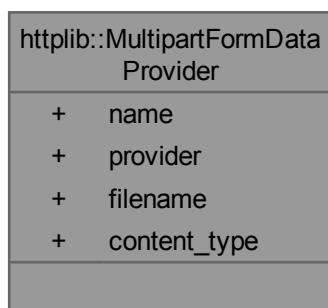
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.24 Структура `httpplib::MultipartFormDataProvider`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::MultipartFormDataProvider`:



Открытые атрибуты

- `std::string name`
- `ContentProviderWithoutLength provider`
- `std::string filename`
- `std::string content_type`

6.24.1 Подробное описание

См. определение в файле `httpplib.h` строка [583](#)

6.24.2 Данные класса

6.24.2.1 `content_type`

`std::string httpplib::MultipartFormDataProvider::content_type`

См. определение в файле `httpplib.h` строка [587](#)

6.24.2.2 `filename`

`std::string httpplib::MultipartFormDataProvider::filename`

См. определение в файле `httpplib.h` строка [586](#)

6.24.2.3 `name`

`std::string httpplib::MultipartFormDataProvider::name`

См. определение в файле `httpplib.h` строка [584](#)

6.24.2.4 `provider`

`ContentProviderWithoutLength httpplib::MultipartFormDataProvider::provider`

См. определение в файле `httpplib.h` строка [585](#)

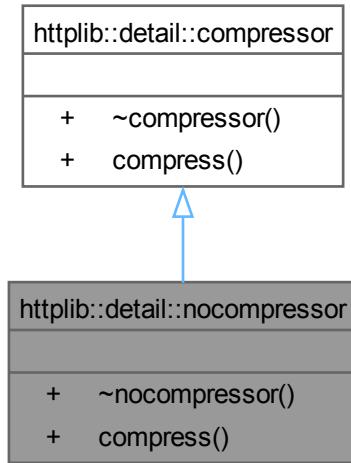
Объявления и описания членов структуры находятся в файле:

- `httpplib.h`

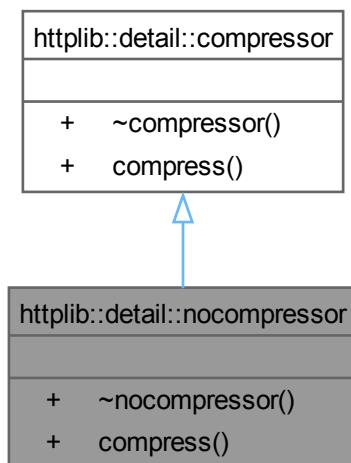
6.25 Класс `httpplib::detail::nocompressor`

```
#include <httpplib.h>
```

Граф наследования:`httpplib::detail::nocompressor`:



Граф связей класса `httpplib::detail::nocompressor`:



Открытые члены

- `~nocompressor ()` override=default
- bool `compress (const char *data, size_t data_length, bool, Callback callback)` override

Открытые члены унаследованные от `httpplib::detail::compressor`

- `virtual ~nocompressor ()=default`

Дополнительные унаследованные члены

Открытые типы унаследованные от `httpplib::detail::compressor`

- `typedef std::function< bool(const char *data, size_t data_len)> Callback`

6.25.1 Подробное описание

См. определение в файле `httpplib.h` строка 2501

6.25.2 Конструктор(ы)

6.25.2.1 `~nocompressor()`

`httpplib::detail::nocompressor::~nocompressor () [override], [default]`

6.25.3 Методы

6.25.3.1 `compress()`

```
bool httpplib::detail::nocompressor::compress (
    const char * data,
    size_t data_length,
    bool ,
    Callback callback) [inline], [override], [virtual]
```

Замещает `httpplib::detail::compressor`.

См. определение в файле `httpplib.h` строка 4002

```
04003 {  
04004 if (!data_length) { return true; }  
04005 return callback(data, data_length);  
04006 }
```

Объявления и описания членов класса находятся в файле:

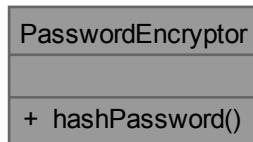
- `httpplib.h`

6.26 Класс PasswordEncryptor

Утилита для безопасного хеширования паролей.

```
#include <PasswordEncryptor.h>
```

Граф связей класса PasswordEncryptor:



Открытые статические члены

- static std::string [hashPassword](#) (const std::string &password)
Хеширует пароль с использованием SHA-256.

6.26.1 Подробное описание

Утилита для безопасного хеширования паролей.

Класс предоставляет простой интерфейс для получения хеш-значения пароля с использованием алгоритма SHA-256. Используется для хранения паролей в базе данных в виде хеша, чтобы предотвратить компрометацию даже при утечке базы данных.

Принцип работы:

- На вход подаётся строка (пароль)
- Производится хеширование по алгоритму SHA-256
- Результатом является строка длиной 64 символа в шестнадцатеричном виде

См. определение в файле [PasswordEncryptor.h](#) строка 17

6.26.2 Методы

6.26.2.1 hashPassword()

```
std::string PasswordEncryptor::hashPassword (
    const std::string & password) [static]
```

Хеширует пароль с использованием SHA-256.

Аргументы

password	Открытый (plaintext) пароль пользователя
----------	------------------------------------------

Возвращает

Строка длиной 64 символов — хеш пароля в hex-формате

См. определение в файле [PasswordEncryptor.cpp](#) строка 4

```
00004
00005     return SHA256::hash(password);
00006 }
```

Граф вызовов:



Граф вызова функции:



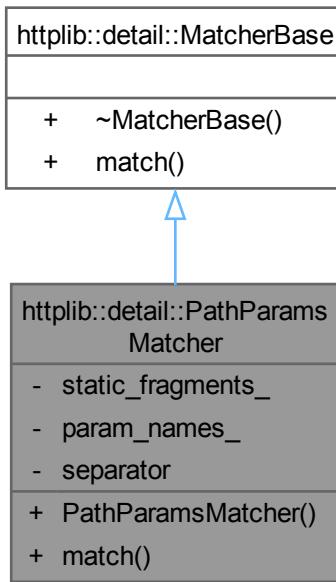
Объявления и описания членов классов находятся в файлах:

- [PasswordEncryptor.h](#)
- [PasswordEncryptor.cpp](#)

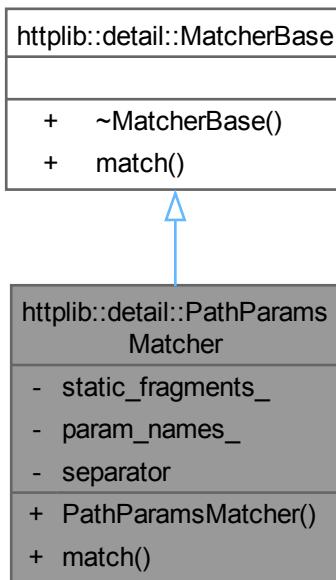
6.27 Класс httpplib::detail::PathParamMatcher

```
#include <httpplib.h>
```

Граф наследования:`httpplib::detail::PathParamsMatcher`:



Граф связей класса `httpplib::detail::PathParamsMatcher`:



Открытые члены

- `PathParamsMatcher` (`const std::string &pattern`)
- `bool match (Request &request) const override`

Открытые члены унаследованные от `httpplib::detail::MatcherBase`

- `virtual ~MatcherBase ()=default`

Закрытые данные

- `std::vector< std::string > static_fragments_`
- `std::vector< std::string > param_names_`

Закрытые статические данные

- `static constexpr char separator = '/'`

6.27.1 Подробное описание

Captures parameters in request path and stores them in `Request::path_params`

Capture name is a substring of a pattern from : to /. The rest of the pattern is matched against the request path directly Parameters are captured starting from the next character after the end of the last matched static pattern fragment until the next /.

Example pattern: "/path/fragments/:capture/more/fragments/:second_capture" Static fragments \leftarrow : "/path/fragments/", "more/fragments/"

Given the following request path: "/path/fragments/:1/more/fragments/:2" the resulting capture will be `{"capture", "1"}, {"second_capture", "2"}`

См. определение в файле `httpplib.h` строка 905

6.27.2 Конструктор(ы)

6.27.2.1 PathParamsMatcher()

```
httpplib::detail::PathParamsMatcher::PathParamsMatcher (
    const std::string & pattern) [inline]
```

См. определение в файле `httpplib.h` строка 6210

```
06210                                     {
06211     constexpr const char marker[] = "/";
06212
06213     // One past the last ending position of a path param substring
06214     std::size_t last_param_end = 0;
06215
06216     #ifndef CPPHTTPPLIB_NO_EXCEPTIONS
06217         // Needed to ensure that parameter names are unique during matcher
06218         // construction
06219         // If exceptions are disabled, only last duplicate path
06220         // parameter will be set
06221     std::unordered_set<std::string> param_name_set;
06222     #endif
```

```

06223
06224     while (true) {
06225         const auto marker_pos = pattern.find(
06226             marker, last_param_end == 0 ? last_param_end : last_param_end - 1);
06227         if (marker_pos == std::string::npos) { break; }
06228
06229         static_fragments_.push_back(
06230             pattern.substr(last_param_end, marker_pos - last_param_end + 1));
06231
06232         const auto param_name_start = marker_pos + str_len(marker);
06233
06234         auto sep_pos = pattern.find(separator, param_name_start);
06235         if (sep_pos == std::string::npos) { sep_pos = pattern.length(); }
06236
06237         auto param_name =
06238             pattern.substr(param_name_start, sep_pos - param_name_start);
06239
06240 #ifndef CPPHTTPPLIB_NO_EXCEPTIONS
06241     if (param_name_set.find(param_name) != param_name_set.cend()) {
06242         std::string msg = "Encountered path parameter '" + param_name +
06243             "' multiple times in route pattern '" + pattern + "'";
06244         throw std::invalid_argument(msg);
06245     }
06246 #endif
06247
06248     param_names_.push_back(std::move(param_name));
06249
06250     last_param_end = sep_pos + 1;
06251 }
06252
06253 if (last_param_end < pattern.length()) {
06254     static_fragments_.push_back(pattern.substr(last_param_end));
06255 }
06256 }
```

Граф вызовов:



6.27.3 Методы

6.27.3.1 `match()`

```
bool httpplib::detail::PathParamsMatcher::match (
    Request & request) const [inline], [override], [virtual]
```

Заменяет `httpplib::detail::MatcherBase`.

См. определение в файле `httpplib.h` строка 6258

```

06258
06259     request.matches = std::smatch();
06260     request.path_params.clear();
06261     request.path_params.reserve(param_names_.size());
06262
06263 // One past the position at which the path matched the pattern last time
06264     std::size_t starting_pos = 0;
06265     for (std::size_t i = 0; i < static_fragments_.size(); ++i) {
06266         const auto &fragment = static_fragments_[i];
06267
06268         if (starting_pos + fragment.length() > request.path.length()) {
06269             return false;
06270         }
06271 }
```

```

06272 // Avoid unnecessary allocation by using strncmp instead of substr +
06273 // comparison
06274 if (std::strncmp(request.path.c_str() + starting_pos, fragment.c_str(),
06275                     fragment.length()) != 0) {
06276     return false;
06277 }
06278 starting_pos += fragment.length();
06279
06280 // Should only happen when we have a static fragment after a param
06281 // Example: '/users/:id/subscriptions'
06282 // The 'subscriptions' fragment here does not have a corresponding param
06283 if (i >= param_names_.size()) { continue; }
06284
06285 auto sep_pos = request.path.find(separator, starting_pos);
06286 if (sep_pos == std::string::npos) { sep_pos = request.path.length(); }
06287
06288 const auto &param_name = param_names_[i];
06289
06290 request.path_params.emplace(
06291     param_name, request.path.substr(starting_pos, sep_pos - starting_pos));
06292
06293 // Mark everything up to '/' as matched
06294 starting_pos = sep_pos + 1;
06295 }
06296
06297 // Returns false if the path is longer than the pattern
06298 return starting_pos >= request.path.length();
06299 }
```

6.27.4 Данные класса

6.27.4.1 `param_names_`

`std::vector<std::string> httpplib::detail::PathParamsMatcher::param_names_ [private]`

См. определение в файле [httpplib.h](#) строка 923

6.27.4.2 `separator`

`char httpplib::detail::PathParamsMatcher::separator = '/' [static], [constexpr], [private]`

См. определение в файле [httpplib.h](#) строка 915

6.27.4.3 `static.fragments_`

`std::vector<std::string> httpplib::detail::PathParamsMatcher::static.fragments_ [private]`

См. определение в файле [httpplib.h](#) строка 920

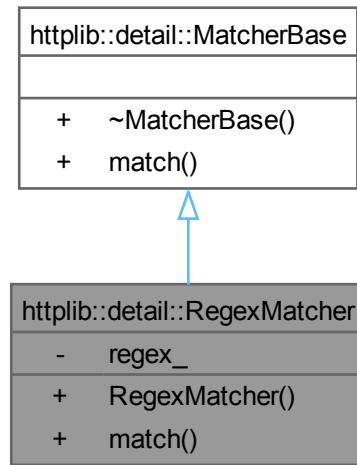
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

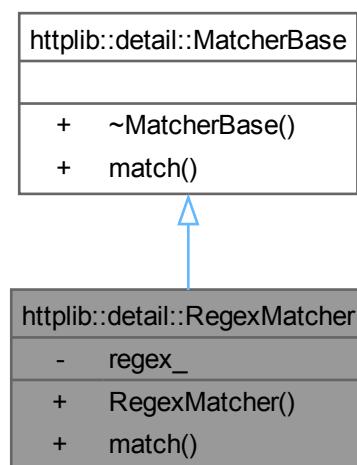
6.28 Класс httpplib::detail::RegexMatcher

```
#include <httpplib.h>
```

Граф наследования: httpplib::detail::RegexMatcher:



Граф связей класса httpplib::detail::RegexMatcher:



Открытые члены

- `RegexMatcher` (const std::string &pattern)
- bool `match` (`Request` &request) const override

Открытые члены унаследованные от `httpplib::detail::MatcherBase`

- `virtual ~MatcherBase ()=default`

Закрытые данные

- `std::regex regex_`

6.28.1 Подробное описание

Performs `std::regex_match` on request path and stores the result in `Request::matches`

Note that regex match is performed directly on the whole request. This means that wildcard patterns may match multiple path segments with `/: "/begin/(.*)/end"` will match both `"/begin/middle/end"` and `"/begin/1/2/end"`.

См. определение в файле `httpplib.h` строка 934

6.28.2 Конструктор(ы)

6.28.2.1 `RegexMatcher()`

```
httpplib::detail::RegexMatcher::RegexMatcher (
    const std::string & pattern) [inline]
```

См. определение в файле `httpplib.h` строка 936
00936 : `regex_(pattern) {}`

6.28.3 Методы

6.28.3.1 `match()`

```
bool httpplib::detail::RegexMatcher::match (
    Request & request) const [inline], [override], [virtual]
```

Замещает `httpplib::detail::MatcherBase`.

См. определение в файле `httpplib.h` строка 6301

```
06301 {
06302     request.path_params.clear();
06303     return std::regex_match(request.path, request.matches, regex_);
06304 }
```

6.28.4 Данные класса

6.28.4.1 `regex_`

```
std::regex httpplib::detail::RegexMatcher::regex_ [private]
```

См. определение в файле `httpplib.h` строка 941

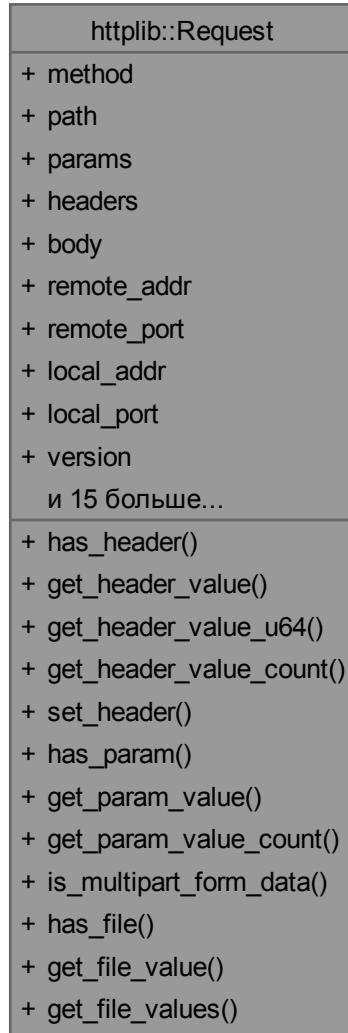
Объявления и описания членов класса находятся в файле:

- `httpplib.h`

6.29 Структура `httpplib::Request`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::Request`:



Открытые члены

- `bool has_header (const std::string &key) const`
- `std::string get_header_value (const std::string &key, const char *def="", size_t id=0) const`
- `uint64_t get_header_value_u64 (const std::string &key, uint64_t def=0, size_t id=0) const`
- `size_t get_header_value_count (const std::string &key) const`
- `void set_header (const std::string &key, const std::string &val)`
- `bool has_param (const std::string &key) const`
- `std::string get_param_value (const std::string &key, size_t id=0) const`

- `size_t get_param_value_count (const std::string &key) const`
- `bool is_multipart_form_data () const`
- `bool has_file (const std::string &key) const`
- `MultipartFormData get_file_value (const std::string &key) const`
- `std::vector< MultipartFormData > get_file_values (const std::string &key) const`

Открытые атрибуты

- `std::string method`
- `std::string path`
- `Params params`
- `Headers headers`
- `std::string body`
- `std::string remote_addr`
- `int remote_port = -1`
- `std::string local_addr`
- `int local_port = -1`
- `std::string version`
- `std::string target`
- `MultipartFormDataMap files`
- `Ranges ranges`
- `Match matches`
- `std::unordered_map< std::string, std::string > path_params`
- `std::function< bool()> is_connection_closed = []() { return true; }`
- `ResponseHandler response_handler`
- `ContentReceiverWithProgress content_receiver`
- `Progress progress`
- `size_t redirect_count_ = 20`
- `size_t content_length_ = 0`
- `ContentProvider content_provider_`
- `bool is_chunked_content_provider_ = false`
- `size_t authorization_count_ = 0`
- `std::chrono::time_point< std::chrono::steady_clock > start_time_`

6.29.1 Подробное описание

См. определение в файле `httpplib.h` строка [627](#)

6.29.2 Методы

6.29.2.1 `get_file_value()`

```
MultipartFormData httpplib::Request::get_file_value (
    const std::string & key) const [inline]
```

См. определение в файле `httpplib.h` строка [5907](#)

```
05907                                         {
05908     auto it = files.find(key);
05909     if (it != files.end()) { return it->second; }
05910     return MultipartFormData();
05911 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.2 `get_file_values()`

```
std::vector< MultipartFormData > httpplib::Request::get_file_values (
    const std::string & key) const [inline]
```

См. определение в файле [httpplib.h](#) строка 5914

```
05914 {
05915     std::vector<MultipartFormData> values;
05916     auto rng = files.equal_range(key);
05917     for (auto it = rng.first; it != rng.second; it++) {
05918         values.push_back(it->second);
05919     }
05920     return values;
05921 }
```

Граф вызовов:



Граф вызова функции:



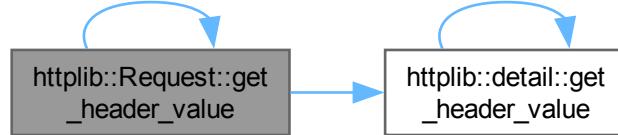
6.29.2.3 `get_header_value()`

```
std::string httpplib::Request::get_header_value (
    const std::string & key,
    const char * def = "",
    size_t id = 0) const [inline]
```

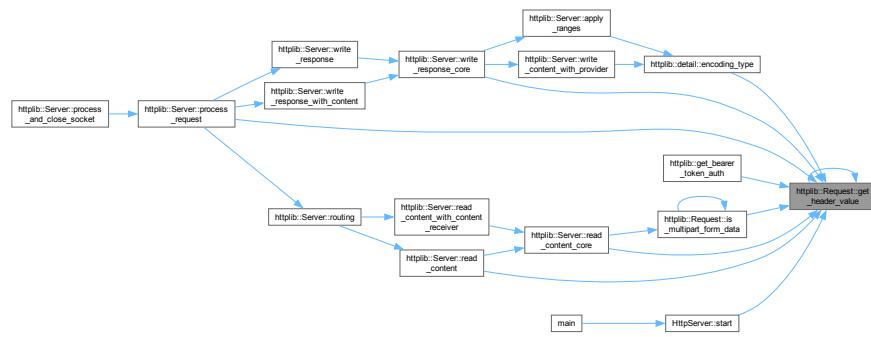
См. определение в файле [httpplib.h](#) строка 5862

```
05863
05864     return detail::get_header_value(headers, key, def, id);
05865 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.4 `get_header_value_count()`

```
size_t httpplib::Request::get_header_value_count (
    const std::string & key) const [inline]
```

См. определение в файле [httpplib.h](#) строка 5867

```
05867                                     {
05868     auto r = headers.equal_range(key);
05869     return static_cast<size_t>(std::distance(r.first, r.second));
05870 }
```

Граф вызовов:



Граф вызова функции:

6.29.2.5 `get_header_value_u64()`

```
uint64_t httpplib::Request::get_header_value_u64 (
    const std::string & key,
    uint64_t def = 0,
    size_t id = 0) const [inline]
```

См. определение в файле [httpplib.h](#) строка 2094

```
02095                                     {
02096     return detail::get_header_value_u64(headers, key, def, id);
02097 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.6 `get_param_value()`

```
std::string httpplib::Request::get_param_value (
    const std::string & key,
    size_t id = 0) const [inline]
```

См. определение в файле [httpplib.h](#) строка [5884](#)

```
05885 {
05886     auto rng = params.equal_range(key);
05887     auto it = rng.first;
05888     std::advance(it, static_cast<ssize_t>(id));
05889     if (it != rng.second) { return it->second; }
05890     return std::string();
05891 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.7 `get_param_value_count()`

```
size_t httpplib::Request::get_param_value_count (
    const std::string & key) const [inline]
```

См. определение в файле `httpplib.h` строка 5893

```
05893     {
05894     auto r = params.equal_range(key);
05895     return static_cast<size_t>(std::distance(r.first, r.second));
05896 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.8 `has_file()`

```
bool httpplib::Request::has_file (
    const std::string & key) const [inline]
См. определение в файле httpplib.h строка 5903
05903 {
05904     return files.find(key) != files.end();
05905 }
```

Граф вызовов:



Граф вызова функции:

6.29.2.9 `has_header()`

```
bool httpplib::Request::has_header (
    const std::string & key) const [inline]
```

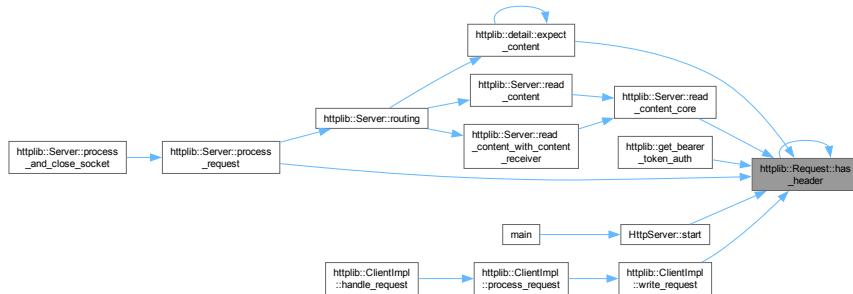
См. определение в файле [httpplib.h](#) строка 5858

```
05858 {
05859     return detail::has_header(headers, key);
05860 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.10 `has_param()`

```
bool httpplib::Request::has_param (
    const std::string & key) const [inline]
```

См. определение в файле `httpplib.h` строка 5880

```
05880 {
05881     return params.find(key) != params.end();
05882 }
```

Граф вызовов:



Граф вызова функции:



6.29.2.11 `is_multipart_form_data()`

```
bool httpplib::Request::is_multipart_form_data () const [inline]
```

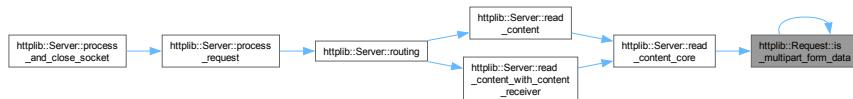
См. определение в файле `httpplib.h` строка 5898

```
05898     {
05899     const auto &content_type = get_header_value("Content-Type");
05900     return !content_type.rfind("multipart/form-data", 0);
05901 }
```

Граф вызовов:



Граф вызова функции:

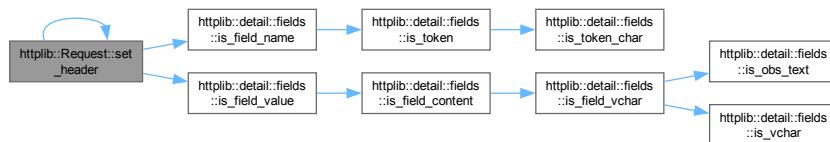
6.29.2.12 `set_header()`

```
void httpplib::Request::set_header (
    const std::string & key,
    const std::string & val) [inline]
```

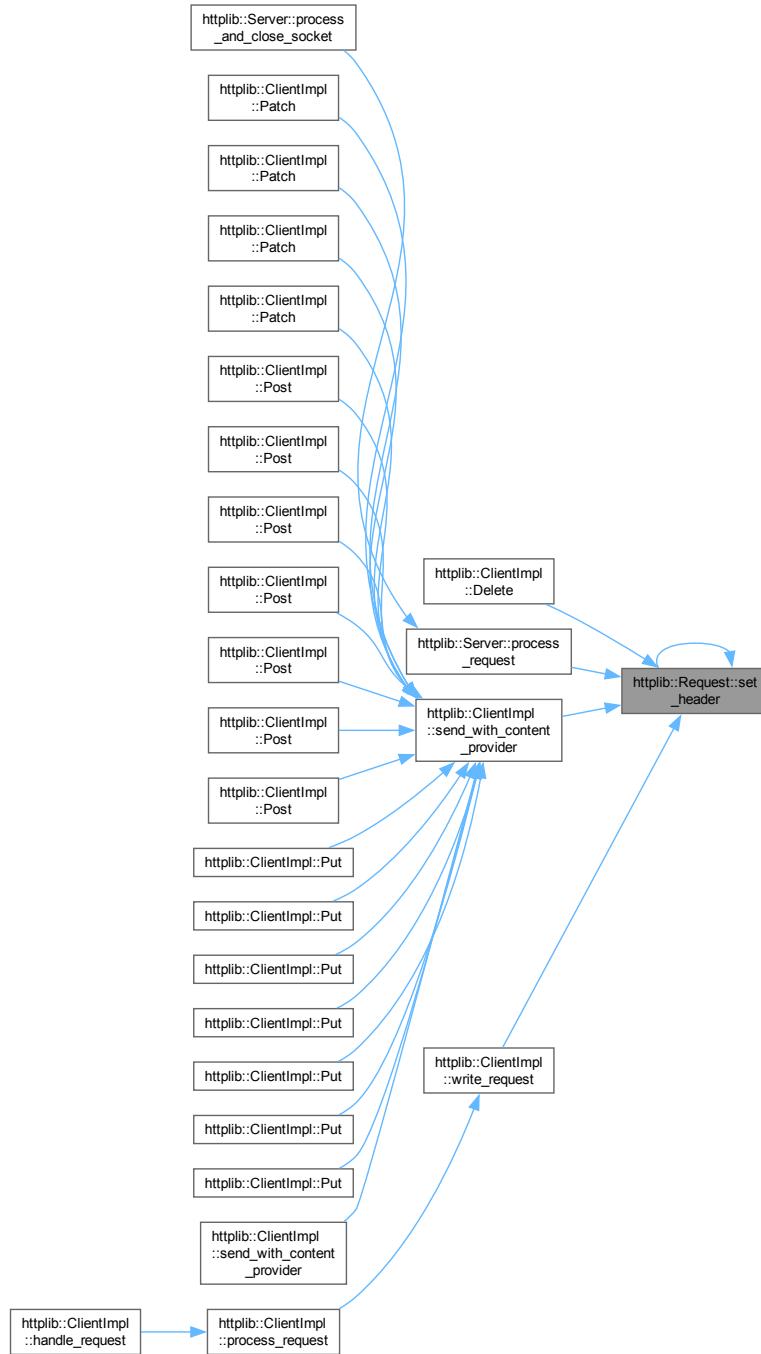
См. определение в файле `httpplib.h` строка 5872

```
05873     {
05874     if (detail::fields::is_field_name(key) &&
05875         detail::fields::is_field_value(val)) {
05876         headers.emplace(key, val);
05877     }
05878 }
```

Граф вызовов:



Граф вызова функций:



6.29.3 Данные класса

6.29.3.1 authorization_count_

```
size_t httpplib::Request::authorization_count_ = 0
```

См. определение в файле [httpplib.h](#) строка 679

6.29.3.2 body

`std::string httpplib::Request::body`

См. определение в файле [httpplib.h](#) строка [632](#)

6.29.3.3 content_length_

`size_t httpplib::Request::content_length_ = 0`

См. определение в файле [httpplib.h](#) строка [676](#)

6.29.3.4 content_provider_

`ContentProvider httpplib::Request::content_provider_`

См. определение в файле [httpplib.h](#) строка [677](#)

6.29.3.5 content_receiver

`ContentReceiverWithProgress httpplib::Request::content_receiver`

См. определение в файле [httpplib.h](#) строка [650](#)

6.29.3.6 files

`MultipartFormDataMap httpplib::Request::files`

См. определение в файле [httpplib.h](#) строка [642](#)

6.29.3.7 headers

`Headers httpplib::Request::headers`

См. определение в файле [httpplib.h](#) строка [631](#)

6.29.3.8 is_chunked_content_provider_

`bool httpplib::Request::is_chunked_content_provider_ = false`

См. определение в файле [httpplib.h](#) строка [678](#)

6.29.3.9 is_connection_closed

`std::function<bool()> httpplib::Request::is_connection_closed = []() { return true; }`

См. определение в файле [httpplib.h](#) строка [646](#)
00646 { `return true;` };

6.29.3.10 `local_addr`

`std::string httpplib::Request::local_addr`

См. определение в файле [httpplib.h](#) строка [636](#)

6.29.3.11 `local_port`

`int httpplib::Request::local_port = -1`

См. определение в файле [httpplib.h](#) строка [637](#)

6.29.3.12 `matches`

`Match httpplib::Request::matches`

См. определение в файле [httpplib.h](#) строка [644](#)

6.29.3.13 `method`

`std::string httpplib::Request::method`

См. определение в файле [httpplib.h](#) строка [628](#)

6.29.3.14 `params`

`Params httpplib::Request::params`

См. определение в файле [httpplib.h](#) строка [630](#)

6.29.3.15 `path`

`std::string httpplib::Request::path`

См. определение в файле [httpplib.h](#) строка [629](#)

6.29.3.16 `path_params`

`std::unordered_map<std::string, std::string> httpplib::Request::path_params`

См. определение в файле [httpplib.h](#) строка [645](#)

6.29.3.17 `progress`

`Progress httpplib::Request::progress`

См. определение в файле [httpplib.h](#) строка [651](#)

6.29.3.18 `ranges`

`Ranges` `httpplib::Request::ranges`

См. определение в файле [httpplib.h](#) строка [643](#)

6.29.3.19 `redirect_count_`

`size_t httpplib::Request::redirect_count_ = 20`

См. определение в файле [httpplib.h](#) строка [675](#)

6.29.3.20 `remote_addr`

`std::string httpplib::Request::remote_addr`

См. определение в файле [httpplib.h](#) строка [634](#)

6.29.3.21 `remote_port`

`int httpplib::Request::remote_port = -1`

См. определение в файле [httpplib.h](#) строка [635](#)

6.29.3.22 `response_handler`

`ResponseHandler httpplib::Request::response_handler`

См. определение в файле [httpplib.h](#) строка [649](#)

6.29.3.23 `start_time_`

`std::chrono::time_point<std::chrono::steady_clock> httpplib::Request::start_time_`

Инициализатор

`= (std::chrono::steady_clock::time_point::min)()`

См. определение в файле [httpplib.h](#) строка [680](#)

6.29.3.24 `target`

`std::string httpplib::Request::target`

См. определение в файле [httpplib.h](#) строка [641](#)

6.29.3.25 `version`

`std::string httpplib::Request::version`

См. определение в файле [httpplib.h](#) строка 640

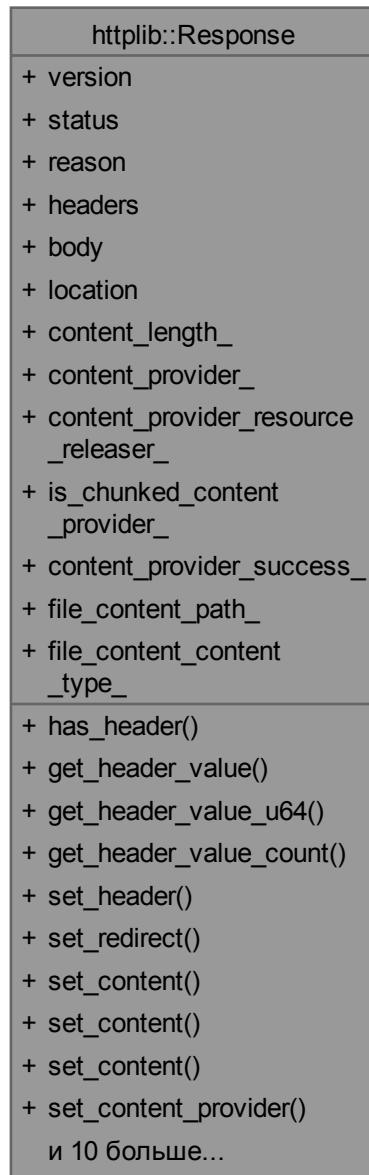
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.30 Структура `httpplib::Response`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::Response`:



Открытые члены

- bool `has_header` (const std::string &key) const
- std::string `get_header_value` (const std::string &key, const char *def="", size_t id=0) const
- uint64_t `get_header_value_u64` (const std::string &key, uint64_t def=0, size_t id=0) const
- size_t `get_header_value_count` (const std::string &key) const
- void `set_header` (const std::string &key, const std::string &val)
- void `set_redirect` (const std::string &url, int `status=StatusCode::Found_302`)
- void `set_content` (const char *s, size_t n, const std::string &content_type)

- `void set_content (const std::string &s, const std::string &content_type)`
- `void set_content (std::string &&s, const std::string &content_type)`
- `void set_content_provider (size_t length, const std::string &content_type, ContentProvider provider, ContentProviderResourceReleaser resource_releaser=nullptr)`
- `void set_content_provider (const std::string &content_type, ContentProviderWithoutLength provider, ContentProviderResourceReleaser resource_releaser=nullptr)`
- `void set_chunked_content_provider (const std::string &content_type, ContentProviderWithoutLength provider, ContentProviderResourceReleaser resource_releaser=nullptr)`
- `void set_file_content (const std::string &path, const std::string &content_type)`
- `void set_file_content (const std::string &path)`
- `Response ()=default`
- `Response (const Response &)=default`
- `Response & operator= (const Response &z)=default`
- `Response (Response &&)=default`
- `Response & operator= (Response &&)=default`
- `~Response ()`

Открытые атрибуты

- `std::string version`
- `int status = -1`
- `std::string reason`
- `Headers headers`
- `std::string body`
- `std::string location`
- `size_t content_length_ = 0`
- `ContentProvider content_provider_`
- `ContentProviderResourceReleaser content_provider_resource_releaser_`
- `bool is_chunked_content_provider_ = false`
- `bool content_provider_success_ = false`
- `std::string file_content_path_`
- `std::string file_content_content_type_`

6.30.1 Подробное описание

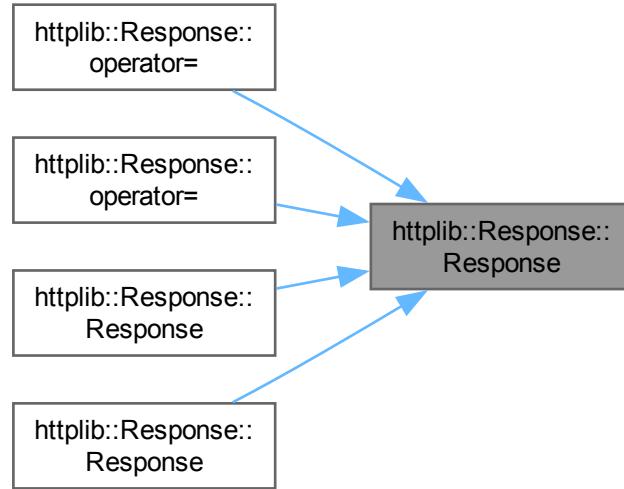
См. определение в файле `httpplib.h` строка 684

6.30.2 Конструктор(ы)

6.30.2.1 `Response()` [1/3]

`httpplib::Response::Response () [default]`

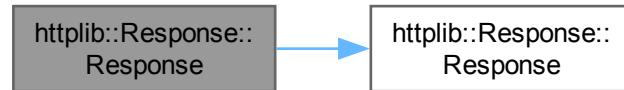
Граф вызова функции:



6.30.2.2 Response() [2/3]

```
httpplib::Response::Response (\n    const Response & ) [default]
```

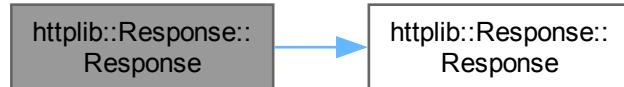
Граф вызовов:



6.30.2.3 Response() [3/3]

```
httpplib::Response::Response (\n    Response && ) [default]
```

Граф вызовов:



6.30.2.4 ~Response()

`httpplib::Response::~Response () [inline]`

См. определение в файле [httpplib.h](#) строка 726

```
00726     {
00727     if (content_provider_resource_releaser_) {
00728         content_provider_resource_releaser_(content_provider_success_);
00729     }
00730 }
```

6.30.3 Методы

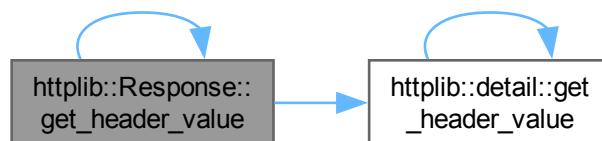
6.30.3.1 get_header_value()

```
std::string httpplib::Response::get_header_value (
    const std::string & key,
    const char * def = "",
    size_t id = 0) const [inline]
```

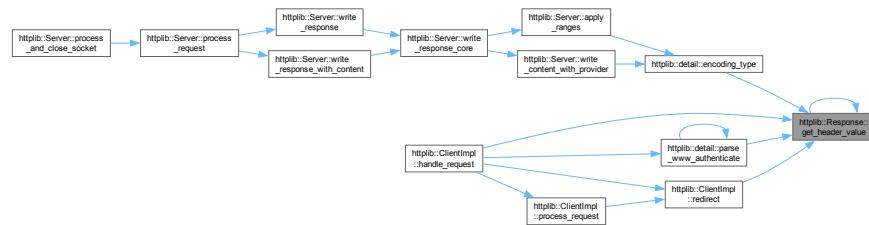
См. определение в файле [httpplib.h](#) строка 5928

```
05930     {
05931     return detail::get_header_value(headers, key, def, id);
05932 }
```

Граф вызовов:



Граф вызова функции:



6.30.3.2 `get_header_value_count()`

```
size_t httpplib::Response::get_header_value_count (
    const std::string & key) const [inline]
```

См. определение в файле [httpplib.h](#) строка 5934

```
05934 {
05935     auto r = headers.equal_range(key);
05936     return static_cast<size_t>(std::distance(r.first, r.second));
05937 }
```

Граф вызовов:



Граф вызова функции:



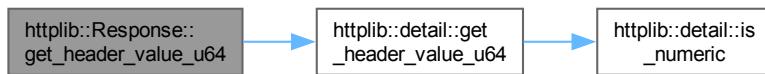
6.30.3.3 `get_header_value_u64()`

```
uint64_t httpplib::Response::get_header_value_u64 (
    const std::string & key,
    uint64_t def = 0,
    size_t id = 0) const [inline]
```

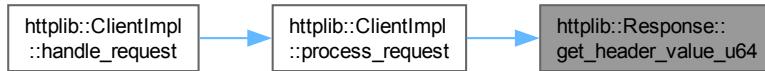
См. определение в файле [httpplib.h](#) строка 2099

```
02100
02101 return detail::get_header_value_u64(headers, key, def, id);
02102 }
```

Граф вызовов:



Граф вызова функции:



6.30.3.4 `has_header()`

```
bool httpplib::Response::has_header (
    const std::string & key) const [inline]
```

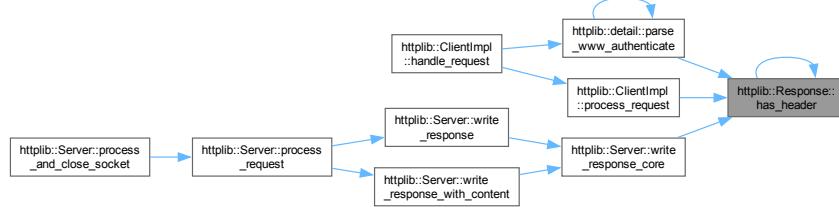
См. определение в файле [httpplib.h](#) строка 5924

```
05924
05925 return headers.find(key) != headers.end();
05926 }
```

Граф вызовов:



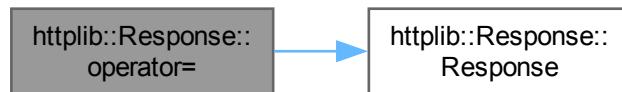
Граф вызова функции:



6.30.3.5 `operator=()` [1/2]

```
Response & http://lib::Response::operator= (
    const Response & ) [default]
```

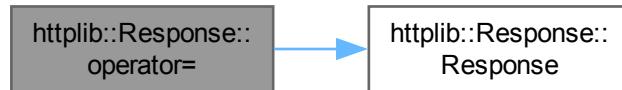
Граф вызовов:



6.30.3.6 `operator=()` [2/2]

```
Response & http://lib::Response::operator= (
    Response && ) [default]
```

Граф вызовов:



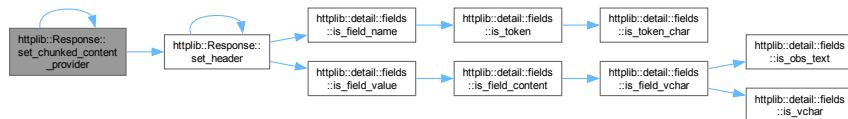
6.30.3.7 `set_chunked_content_provider()`

```
void httpplib::Response::set_chunked_content_provider (
    const std::string & content_type,
    ContentProviderWithoutLength provider,
    ContentProviderResourceReleaser resource_releaser = nullptr) [inline]
```

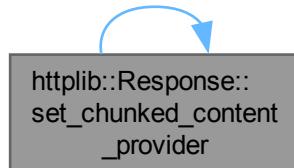
См. определение в файле [httpplib.h](#) строка 6001

```
06003     {
06004     set_header("Content-Type", content_type);
06005     content_length_ = 0;
06006     content_provider_ = detail::ContentProviderAdapter(std::move(provider));
06007     content_provider_resource_releaser_ = std::move(resource_releaser);
06008     is_chunked_content_provider_ = true;
06009 }
```

Граф вызовов:



Граф вызова функции:

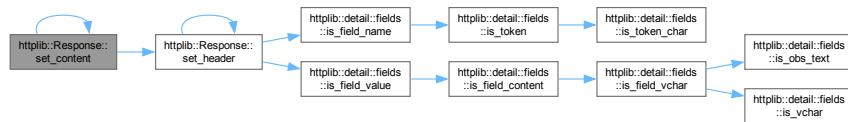
6.30.3.8 `set_content()` [1/3]

```
void httpplib::Response::set_content (
    const char * s,
    size_t n,
    const std::string & content_type) [inline]
```

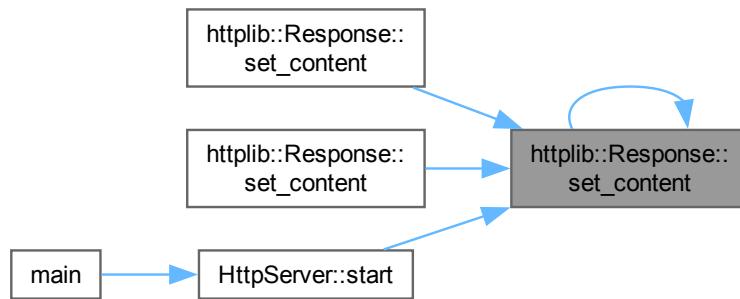
См. определение в файле [httpplib.h](#) строка 5958

```
05959     {
05960     body.assign(s, n);
05961
05962     auto rng = headers.equal_range("Content-Type");
05963     headers.erase(rng.first, rng.second);
05964     set_header("Content-Type", content_type);
05965 }
```

Граф вызовов:



Граф вызова функции:



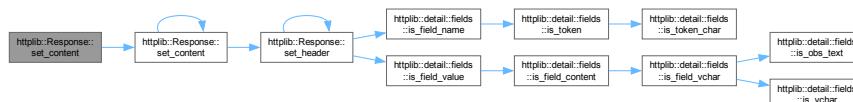
6.30.3.9 set_content() [2/3]

```
void httpplib::Response::set_content (
    const std::string & s,
    const std::string & content_type) [inline]
```

См. определение в файле [httpplib.h](#) строка 5967

```
05968
05969   set_content(s.data(), s.size(), content_type);
05970 }
```

Граф вызовов:



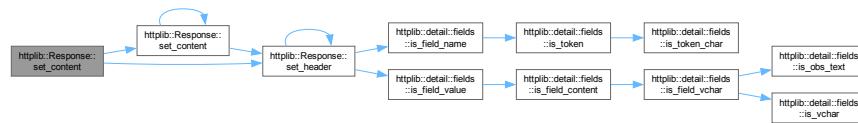
6.30.3.10 `set_content()` [3/3]

```
void httpplib::Response::set_content (
    std::string && s,
    const std::string & content_type) [inline]
```

См. определение в файле [httpplib.h](#) строка 5972

```
05973                                         {
05974     body = std::move(s);
05975
05976     auto rng = headers.equal_range("Content-Type");
05977     headers.erase(rng.first, rng.second);
05978     set_header("Content-Type", content_type);
05979 }
```

Граф вызовов:

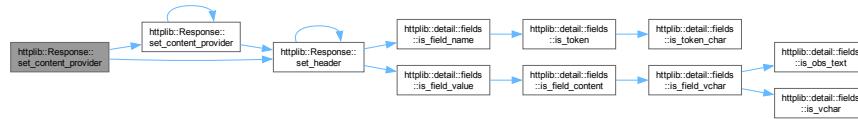
6.30.3.11 `set_content_provider()` [1/2]

```
void httpplib::Response::set_content_provider (
    const std::string & content_type,
    ContentProviderWithoutLength provider,
    ContentProviderResourceReleaser resource_releaser = nullptr) [inline]
```

См. определение в файле [httpplib.h](#) строка 5991

```
05993                                         {
05994     set_header("Content-Type", content_type);
05995     content_length_ = 0;
05996     content_provider_ = detail::ContentProviderAdapter(std::move(provider));
05997     content_provider_resource_releaser_ = std::move(resource_releaser);
05998     is_chunked_content_provider_ = false;
05999 }
```

Граф вызовов:



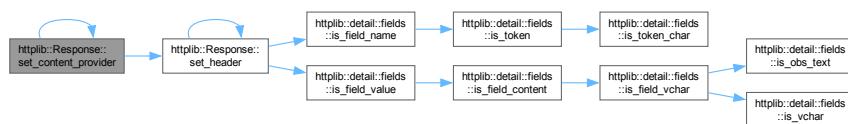
6.30.3.12 `set_content_provider()` [2/2]

```
void httpplib::Response::set_content_provider (
    size_t length,
    const std::string & content_type,
    ContentProvider provider,
    ContentProviderResourceReleaser resource_releaser = nullptr) [inline]
```

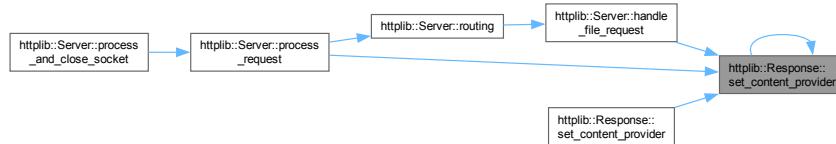
См. определение в файле `httpplib.h` строка 5981

```
05983     {
05984     set_header("Content-Type", content_type);
05985     content_length_ = in_length;
05986     if (in_length > 0) { content_provider_ = std::move(provider); }
05987     content_provider_resource_releaser_ = std::move(resource_releaser);
05988     is_chunked_content_provider_ = false;
05989 }
```

Граф вызовов:



Граф вызова функции:

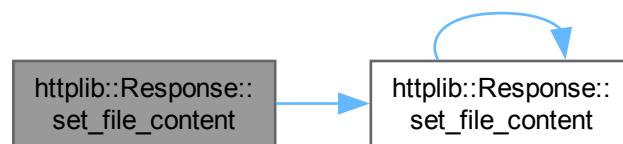
6.30.3.13 `set_file_content()` [1/2]

```
void httpplib::Response::set_file_content (
    const std::string & path) [inline]
```

См. определение в файле `httpplib.h` строка 6017

```
06017
06018     file_content_path_ = path;
06019 }
```

Граф вызовов:



6.30.3.14 `set_file_content()` [2/2]

```
void httpplib::Response::set_file_content (
    const std::string & path,
    const std::string & content_type) [inline]
```

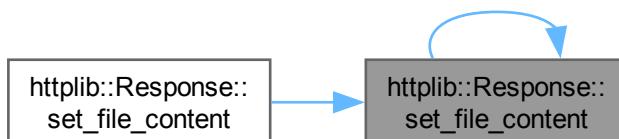
См. определение в файле [httpplib.h](#) строка 6011

```
06012 {
06013     file_content_path_ = path;
06014     file_content_content_type_ = content_type;
06015 }
```

Граф вызовов:



Граф вызова функции:

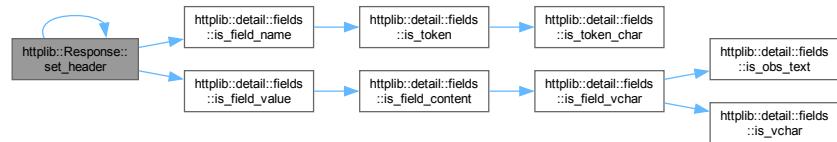
6.30.3.15 `set_header()`

```
void httpplib::Response::set_header (
    const std::string & key,
    const std::string & val) [inline]
```

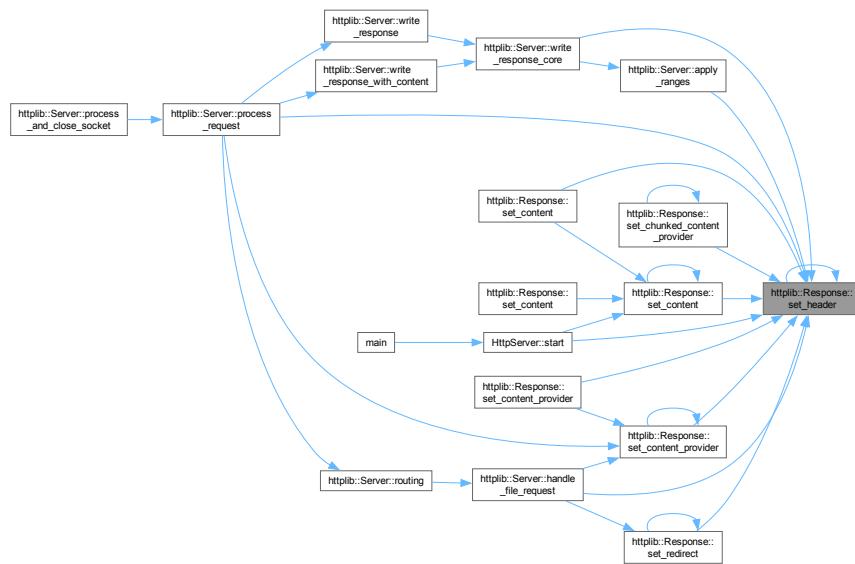
См. определение в файле [httpplib.h](#) строка 5939

```
05940     {
05941         if (detail::fields::is_field_name(key) &&
05942             detail::fields::is_field_value(val)) {
05943             headers.emplace(key, val);
05944         }
05945     }
```

Граф вызовов:



Граф вызова функции:



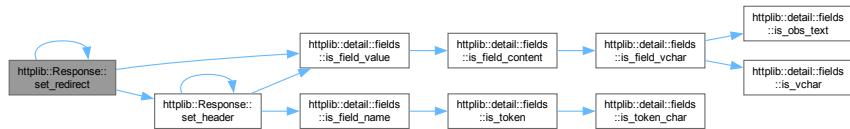
6.30.3.16 `set_redirect()`

```
void httpplib::Response::set_redirect (
    const std::string & url,
    int status = StatusCode::Found_302) [inline]
```

См. определение в файле `httpplib.h` строка 5947

```
05947
05948 if (detail::fields::is_field_value(url)) {
05949     set_header("Location", url);
05950     if (300 <= stat && stat < 400) {
05951         this->status = stat;
05952     } else {
05953         this->status = StatusCode::Found_302;
05954     }
05955 }
05956 }
```

Граф вызовов:



Граф вызова функций:



6.30.4 Данные класса

6.30.4.1 body

`std::string httpplib::Response::body`

См. определение в файле [httpplib.h](#) строка 689

6.30.4.2 content_length_

`size_t httpplib::Response::content_length_ = 0`

См. определение в файле [httpplib.h](#) строка 733

6.30.4.3 content_provider_

`ContentProvider httpplib::Response::content_provider_`

См. определение в файле [httpplib.h](#) строка 734

6.30.4.4 content_provider_resource_releaser_

`ContentProviderResourceReleaser httpplib::Response::content_provider_resource_releaser_`

См. определение в файле [httpplib.h](#) строка 735

6.30.4.5 content_provider_success_

`bool httpplib::Response::content_provider_success_ = false`

См. определение в файле [httpplib.h](#) строка 737

6.30.4.6 `file_content_content_type_`

`std::string httpplib::Response::file_content_content_type_`

См. определение в файле [httpplib.h](#) строка [739](#)

6.30.4.7 `file_content_path_`

`std::string httpplib::Response::file_content_path_`

См. определение в файле [httpplib.h](#) строка [738](#)

6.30.4.8 `headers`

[Headers](#) `httpplib::Response::headers`

См. определение в файле [httpplib.h](#) строка [688](#)

6.30.4.9 `is_chunked_content_provider_`

`bool httpplib::Response::is_chunked_content_provider_ = false`

См. определение в файле [httpplib.h](#) строка [736](#)

6.30.4.10 `location`

`std::string httpplib::Response::location`

См. определение в файле [httpplib.h](#) строка [690](#)

6.30.4.11 `reason`

`std::string httpplib::Response::reason`

См. определение в файле [httpplib.h](#) строка [687](#)

6.30.4.12 `status`

`int httpplib::Response::status = -1`

См. определение в файле [httpplib.h](#) строка [686](#)

6.30.4.13 version

std::string httpplib::Response::version

См. определение в файле [httpplib.h](#) строка 685

Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.31 Класс httpplib::Result

#include <httpplib.h>

Граф связей класса httpplib::Result:

httpplib::Result	
-	res_
-	err_
-	request_headers_
+	Result()
+	Result()
+	operator bool()
+	operator==()
+	operator!=()
+	value()
+	value()
+	operator*()
+	operator*()
+	operator->()
и 6 больше...	

Открытые члены

- [Result \(\)=default](#)
- [Result \(std::unique_ptr< Response > &&res, Error err, Headers &&request_headers=Headers{}\)](#)
- [operator bool \(\) const](#)
- [bool operator==\(std::nullptr_t \) const](#)
- [bool operator!=\(std::nullptr_t \) const](#)
- [const Response & value \(\) const](#)

- `Response & value ()`
- `const Response & operator* () const`
- `Response & operator* ()`
- `const Response * operator-> () const`
- `Response * operator-> ()`
- `Error error () const`
- `bool has_request_header (const std::string &key) const`
- `std::string get_request_header_value (const std::string &key, const char *def="", size_t id=0) const`
- `uint64_t get_request_header_value_u64 (const std::string &key, uint64_t def=0, size_t id=0) const`
- `size_t get_request_header_value_count (const std::string &key) const`

Закрытые данные

- `std::unique_ptr< Response > res_`
- `Error err_ = Error::Unknown`
- `Headers request_headers_`

6.31.1 Подробное описание

См. определение в файле `httpplib.h` строка 1188

6.31.2 Конструктор(ы)

6.31.2.1 `Result() [1/2]`

`httpplib::Result::Result () [default]`

6.31.2.2 `Result() [2/2]`

```
httpplib::Result::Result (
    std::unique_ptr< Response > && res,
    Error err,
    Headers && request_headers = Headers{}) [inline]
```

См. определение в файле `httpplib.h` строка 1191

```
01192     {}
01193     : res_(std::move(res)), err_(err),
01194       request_headers_(std::move(request_headers)) {}
```

6.31.3 Методы

6.31.3.1 `error()`

`Error httpplib::Result::error () const [inline]`

См. определение в файле `httpplib.h` строка 1207

```
01207 { return err_; }
```

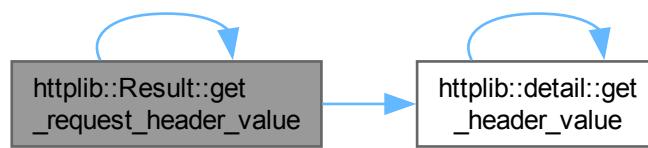
6.31.3.2 `get_request_header_value()`

```
std::string httpplib::Result::get_request_header_value (
    const std::string & key,
    const char * def = "",
    size_t id = 0) const [inline]
```

См. определение в файле [httpplib.h](#) строка 6026

```
06028
06029     return detail::get_header_value(request_headers_, key, def, id);
06030 }
```

Граф вызовов:



Граф вызова функции:

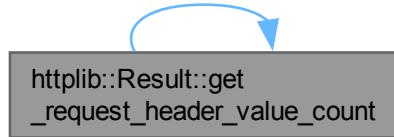
6.31.3.3 `get_request_header_value_count()`

```
size_t httpplib::Result::get_request_header_value_count (
    const std::string & key) const [inline]
```

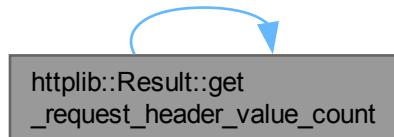
См. определение в файле [httpplib.h](#) строка 6033

```
06033
06034     auto r = request_headers_.equal_range(key);
06035     return static_cast<size_t>(std::distance(r.first, r.second));
06036 }
```

Граф вызовов:



Граф вызова функции:



6.31.3.4 `get_request_header_value_u64()`

```
uint64_t httpplib::Result::get_request_header_value_u64 (
    const std::string & key,
    uint64_t def = 0,
    size_t id = 0) const [inline]
```

См. определение в файле [httpplib.h](#) строка 2286

```
02288 {
02289     return detail::get_header_value_u64(request_headers_, key, def, id);
02290 }
```

Граф вызовов:



6.31.3.5 `has_request_header()`

```
bool httpplib::Result::has_request_header (
    const std::string & key) const [inline]
```

См. определение в файле [httpplib.h](#) строка 6022

```
06022 {  
06023     return request_headers_.find(key) != request_headers_.end();  
06024 }
```

Граф вызовов:



Граф вызова функции:

6.31.3.6 `operator bool()`

```
httpplib::Result::operator bool () const [inline]
```

См. определение в файле [httpplib.h](#) строка 1196

```
01196 { return res_ != nullptr; }
```

6.31.3.7 `operator"!=()`

```
bool httpplib::Result::operator!= (
    std::nullptr_t ) const [inline]
```

См. определение в файле [httpplib.h](#) строка 1198

```
01198 { return res_ != nullptr; }
```

6.31.3.8 `operator*()` [1/2]

```
Response & httpplib::Result::operator* () [inline]
```

См. определение в файле `httpplib.h` строка 1202

```
01202 { return *res_; }
```

6.31.3.9 `operator*()` [2/2]

```
const Response & httpplib::Result::operator* () const [inline]
```

См. определение в файле `httpplib.h` строка 1201

```
01201 { return *res_; }
```

6.31.3.10 `operator->()` [1/2]

```
Response * httpplib::Result::operator-> () [inline]
```

См. определение в файле `httpplib.h` строка 1204

```
01204 { return res_.get(); }
```

6.31.3.11 `operator->()` [2/2]

```
const Response * httpplib::Result::operator-> () const [inline]
```

См. определение в файле `httpplib.h` строка 1203

```
01203 { return res_.get(); }
```

6.31.3.12 `operator==()`

```
bool httpplib::Result::operator== ( std::nullptr_t ) const [inline]
```

См. определение в файле `httpplib.h` строка 1197

```
01197 { return res_ == nullptr; }
```

6.31.3.13 `value()` [1/2]

```
Response & httpplib::Result::value () [inline]
```

См. определение в файле `httpplib.h` строка 1200

```
01200 { return *res_; }
```

6.31.3.14 `value()` [2/2]

```
const Response & httpplib::Result::value () const [inline]
```

См. определение в файле `httpplib.h` строка 1199

```
01199 { return *res_; }
```

6.31.4 Данные класса

6.31.4.1 err_

`Error` `httpplib::Result::err_ = Error::Unknown` [private]

См. определение в файле [httpplib.h](#) строка [1220](#)

6.31.4.2 request_headers_

`Headers` `httpplib::Result::request_headers_` [private]

См. определение в файле [httpplib.h](#) строка [1221](#)

6.31.4.3 res_

`std::unique_ptr<Response>` `httpplib::Result::res_` [private]

См. определение в файле [httpplib.h](#) строка [1219](#)

Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.32 Класс RSA

Класс, реализующий основные операции алгоритма [RSA](#).

```
#include <RSA.h>
```

Граф связей класса RSA:

RSA
+ generate_keys()
+ encrypt()
+ decrypt()
+ sign()
+ verify()

Открытые статические члены

- static void `generate_keys` (`RSAPublicKey` &pub, `RSAPrivateKey` &priv, int bit_length=64)
Генерация пары RSA-ключей.
- static `BigInt encrypt` (const `BigInt` &message, const `RSAPublicKey` &key)
Шифрует сообщение с использованием открытого ключа.
- static `BigInt decrypt` (const `BigInt` &cipher, const `RSAPrivateKey` &key)
Расшифровывает сообщение с использованием приватного ключа.
- static `BigInt sign` (const `BigInt` &hash, const `RSAPrivateKey` &key)
Создаёт цифровую подпись хеша сообщения.
- static bool `verify` (const std::string &messageHashHex, const `BigInt` &signature, const `RSAPublicKey` &key)
Проверяет цифровую подпись по хешу сообщения.

6.32.1 Подробное описание

Класс, реализующий основные операции алгоритма RSA.

Включает в себя генерацию ключей, шифрование/десифрование, подпись и верификацию.

См. определение в файле `RSA.h` строка 33

6.32.2 Методы

6.32.2.1 decrypt()

```
BigInt RSA::decrypt (
    const BigInt & cipher,
    const RSAPrivateKey & key) [static]
```

Расшифровывает сообщение с использованием приватного ключа.

Применяется формула: message = cipher^d mod n.

Аргументы

cipher	Зашифрованное сообщение
key	Приватный ключ

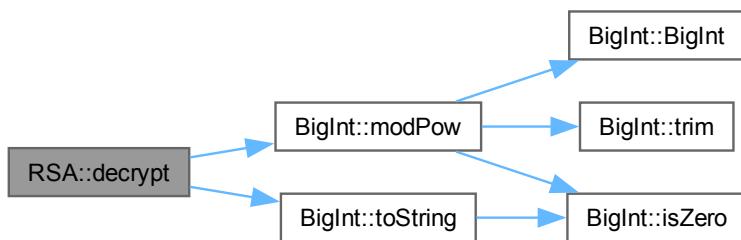
Возвращает

Расшифрованное сообщение

См. определение в файле [RSA.cpp](#) строка 120

```
00120     {
00121     std::cout << "[RSA] --- Расшифровка ---" << std::endl;
00122     std::cout << "Cipher: " << cipher.toString(16) << std::endl;
00123     BigInt message = BigInt::modPow(cipher, key.d, key.n);
00124     std::cout << "Decrypted: " << message.toString(16) << std::endl;
00125     return message;
00126 }
```

Граф вызовов:



6.32.2.2 encrypt()

```
BigInt RSA::encrypt (
    const BigInt & message,
    const RSAPublicKey & key) [static]
```

Шифрует сообщение с использованием открытого ключа.

Применяется формула: cipher = message^e mod n.

Аргументы

message	Открытое сообщение
key	Открытый ключ

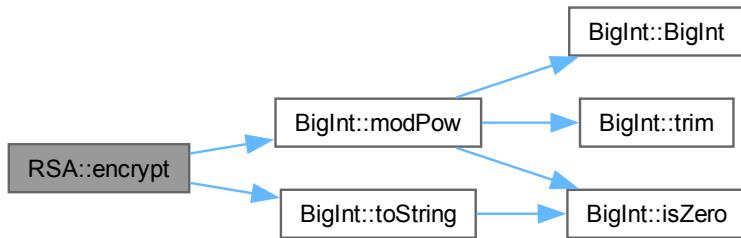
Возвращает

Зашифрованное сообщение

См. определение в файле [RSA.cpp](#) строка 112

```
00112     {
00113     std::cout << "[RSA] --- Шифрование ---" << std::endl;
00114     std::cout << "Message: " << message.toString(16) << std::endl;
00115     BigInt cipher = BigInt::modPow(message, key.e, key.n);
00116     std::cout << "Encrypted: " << cipher.toString(16) << std::endl;
00117     return cipher;
00118 }
```

Граф вызовов:



6.32.2.3 generate_keys()

```
void RSA::generate_keys (
    RSAPublicKey & pub,
    RSAPrivateKey & priv,
    int bit_length = 64) [static]
```

Генерация пары RSA-ключей.

Алгоритм:

1. Выбираются два больших случайных простых числа p и q .
2. Вычисляется $n = p * q$.
3. Вычисляется $\phi(n) = (p - 1) * (q - 1)$.
4. Выбирается публичная экспонента e , обычно 65537, такая, что $\gcd(e, \phi) == 1$.
5. Вычисляется приватная экспонента d , такая что $d * e \equiv 1 \pmod{\phi(n)}$ — обратное по модулю.

Аргументы

out	pub	Структура, в которую будет записан открытый ключ
out	priv	Структура, в которую будет записан приватный ключ
	bit_length	Размер случайных простых чисел (по умолчанию 64 бита)

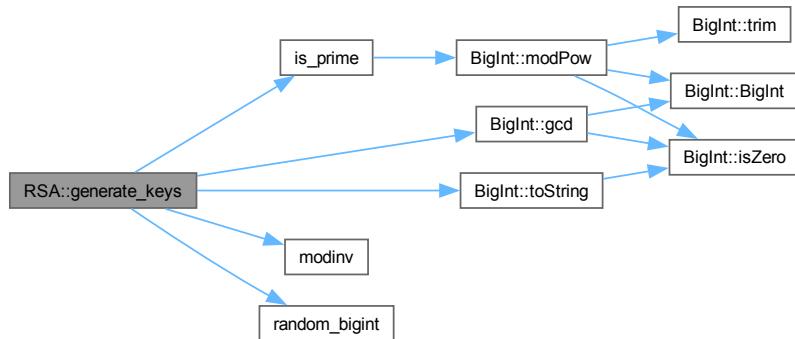
См. определение в файле [RSA.cpp](#) строка 73

```
00073     std::cout << "[RSA] --- Генерация ключей ---" << std::endl;
00074
00075
00076     BigInt p, q;
00077
00078     std::cout << "[RSA] Генерация простого p..." << std::endl;
00079     do {
00080         p = random_bigint(bit_length);
00081     } while (!is_prime(p));
00082     std::cout << "[RSA] Простое p: " << p.toString() << std::endl;
00083
00084     std::cout << "[RSA] Генерация простого q..." << std::endl;
00085     do {
```

```

00086     q = random_bigint(bit_length);
00087 } while (!is_prime(q) || p == q);
00088 std::cout << "[RSA] Простое q: " << q.toString() << std::endl;
00089
00090 BigInt n = p * q;
00091 BigInt phi = (p - BigInt(1)) * (q - BigInt(1));
00092 BigInt e = 65537;
00093
00094 std::cout << "[RSA] n = p * q = " << n.toString() << std::endl;
00095 std::cout << "[RSA] phi = (p-1)*(q-1) = " << phi.toString() << std::endl;
00096
00097 while (BigInt::gcd(e, phi) != BigInt(1)) {
00098     e = e + BigInt(2);
00099 }
00100
00101 std::cout << "[RSA] Публичная экспонента e: " << e.toString() << std::endl;
00102
00103 BigInt d = modinv(e, phi);
00104 std::cout << "[RSA] Приватная экспонента d: " << d.toString() << std::endl;
00105
00106 pub = {e, n};
00107 priv = {d, n};
00108
00109 std::cout << "[RSA] --- Ключи успешно сгенерированы ---" << std::endl;
00110 }
```

Граф вызовов:



Граф вызова функции:



6.32.2.4 sign()

```

BigInt RSA::sign (
    const BigInt & hash,
    const RSAPrivateKey & key) [static]
```

Создаёт цифровую подпись хеша сообщения.

Применяется формула: signature = hash^d mod n.

Аргументы

hash	Хеш сообщения, представленный как число
key	Приватный ключ

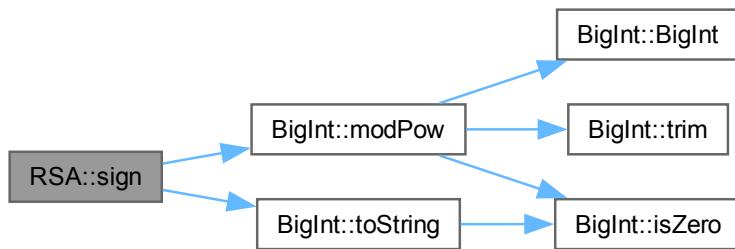
Возвращает

Подпись сообщения

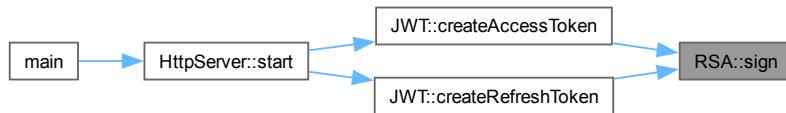
См. определение в файле [RSA.cpp](#) строка 128

```
00128     {
00129     std::cout << "[RSA] --- Подпись ---" << std::endl;
00130     std::cout << "Hash (hex): " << hash.toHexString(16) << std::endl;
00131     BigInt sig = BigInt::modPow(hash, key.d, key.n);
00132     std::cout << "Signature (hex): " << sig.toHexString(16) << std::endl;
00133     return sig;
00134 }
```

Граф вызовов:



Граф вызова функции:



6.32.2.5 verify()

```
bool RSA::verify (
    const std::string & messageHashHex,
    const BigInt & signature,
    const RSApublicKey & key) [static]
```

Проверяет цифровую подпись по хешу сообщения.

Выполняется:

1. 'hash' = $\text{signature}^e \bmod n$
2. Сравнение $\text{hash}' == \text{hash}$, где hash' — это ожидаемый хеш в шестнадцатеричном виде.

Аргументы

messageHashHex	Ожидаемый хеш сообщения (в hex)
signature	Подпись
key	Открытый ключ

Возвращает

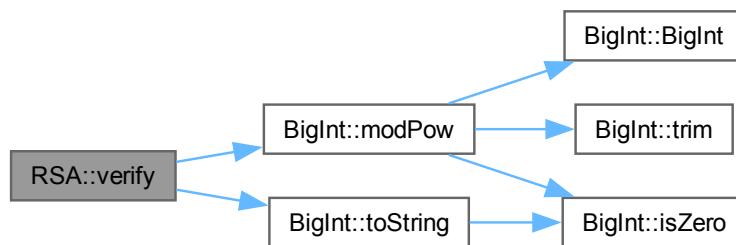
true, если подпись валидна; false — иначе

См. определение в файле [RSA.cpp](#) строка 136

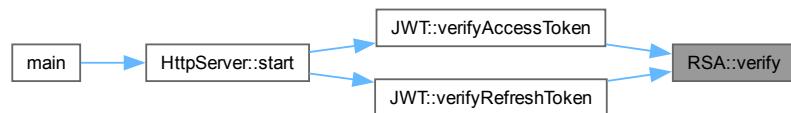
```

00136
00137     std::cout << "[RSA] --- Верификация подписи ---" << std::endl;
00138     std::cout << "Expected hash: " << messageHashHex << std::endl;
00139     std::cout << "Signature:      " << signature.toString(16) << std::endl;
00140
00141     BigInt decryptedHashInt = BigInt::modPow(signature, key.e, key.n);
00142     std::string decryptedHashHex = decryptedHashInt.toString(16);
00143
00144     while (decryptedHashHex.length() < 64)
00145         decryptedHashHex = "0" + decryptedHashHex;
00146
00147     std::cout << "Decrypted hash: " << decryptedHashHex << std::endl;
00148
00149     bool valid = (decryptedHashHex == messageHashHex);
00150     std::cout << "Signature valid: " << (valid ? "YES" : "NO") << std::endl;
00151
00152     return valid;
00153 }
```

Граф вызовов:



Граф вызова функции:



Объявления и описания членов классов находятся в файлах:

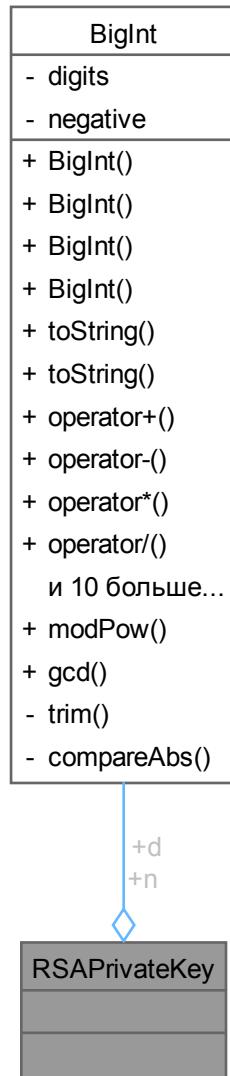
- RSA.h
- RSA.cpp

6.33 Структура RSAPrivateKey

Структура для хранения закрытого (приватного) ключа RSA.

```
#include <RSA.h>
```

Граф связей класса RSAPrivateKey:



Открытые атрибуты

- [BigInt d](#)
Приватная экспонента
- [BigInt n](#)
Модуль

6.33.1 Подробное описание

Структура для хранения закрытого (приватного) ключа [RSA](#).

Приватный ключ состоит из:

- d — приватная экспонента;
- n — тот же модуль, что и в открытом ключе.

См. определение в файле [RSA.h](#) строка 23

6.33.2 Данные класса

6.33.2.1 d

[BigInt](#) RSAPrivatekey::d

Приватная экспонента

См. определение в файле [RSA.h](#) строка 24

6.33.2.2 n

[BigInt](#) RSAPrivatekey::n

Модуль

См. определение в файле [RSA.h](#) строка 25

Объявления и описания членов структуры находятся в файле:

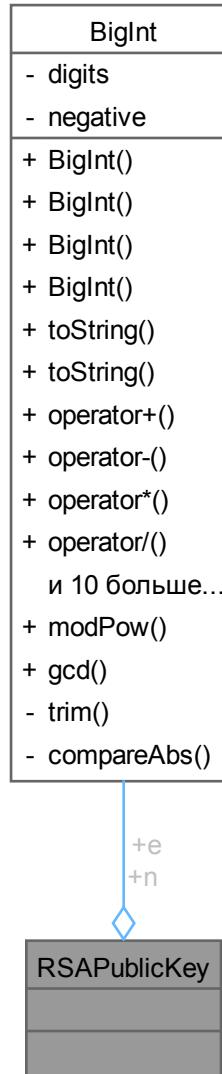
- [RSA.h](#)

6.34 Структура RSAPublicKey

Структура для хранения открытого (публичного) ключа RSA.

```
#include <RSA.h>
```

Граф связей класса RSAPublicKey:



Открытые атрибуты

- **BigInt e**
Публичная экспонента
- **BigInt n**
Модуль

6.34.1 Подробное описание

Структура для хранения открытого (публичного) ключа [RSA](#).

Открытый ключ состоит из:

- `e` — публичная экспонента;
- `n` — модуль (произведение двух больших простых чисел `p` и `q`).

См. определение в файле [RSA.h](#) строка 11

6.34.2 Данные класса

6.34.2.1 `e`

[BigInt](#) RSAPublicKey::`e`

Публичная экспонента

См. определение в файле [RSA.h](#) строка 12

6.34.2.2 `n`

[BigInt](#) RSAPublicKey::`n`

Модуль

См. определение в файле [RSA.h](#) строка 13

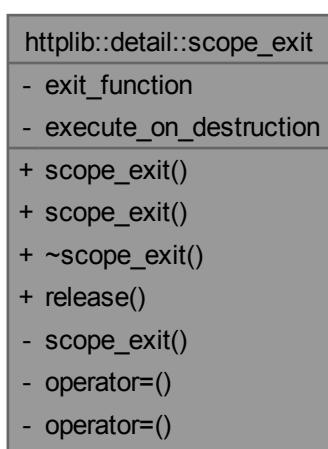
Объявления и описания членов структуры находятся в файле:

- [RSA.h](#)

6.35 Структура `httpplib::detail::scope_exit`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::scope_exit`:



Открытые члены

- `scope_exit` (`std::function< void(void)> &&f)`
- `scope_exit (scope_exit &&rhs)` noexcept
- `~scope_exit ()`
- `void release ()`

Закрытые члены

- `scope_exit (const scope_exit &) = delete`
- `void operator= (const scope_exit &) = delete`
- `scope_exit & operator= (scope_exit &&) = delete`

Закрытые данные

- `std::function< void(void)> exit_function`
- `bool execute_on_destruction`

6.35.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 410

6.35.2 Конструктор(ы)

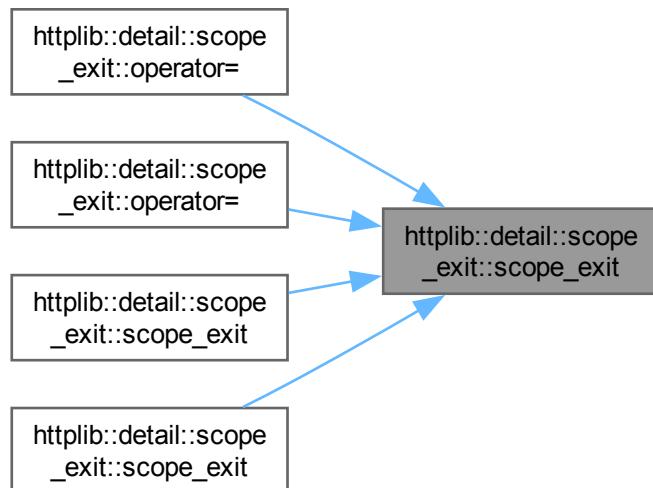
6.35.2.1 `scope_exit()` [1/3]

```
httpplib::detail::scope_exit::scope_exit (
    std::function< void(void)> && f) [inline], [explicit]
```

См. определение в файле [httpplib.h](#) строка 411

```
00412     : exit_function(std::move(f)), execute_on_destruction{true} {}
```

Граф вызова функции:



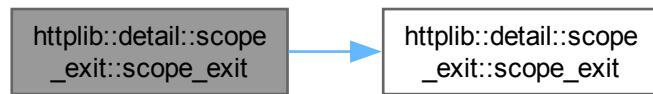
6.35.2.2 `scope_exit()` [2/3]

```
httpplib::detail::scope_exit::scope_exit (
    scope_exit && rhs) [inline], [noexcept]
```

См. определение в файле [httpplib.h](#) строка 414

```
00415     : exit_function(std::move(rhs.exit_function)),
00416         execute_on_destruction{rhs.execute_on_destruction} {
00417     rhs.release();
00418 }
```

Граф вызовов:

6.35.2.3 `~scope_exit()`

```
httpplib::detail::scope_exit::~scope_exit () [inline]
```

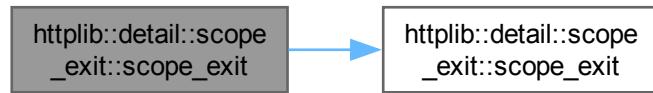
См. определение в файле [httpplib.h](#) строка 420

```
00420     {
00421     if (execute_on_destruction) { this->exit_function(); }
00422 }
```

6.35.2.4 `scope_exit()` [3/3]

```
httpplib::detail::scope_exit::scope_exit (
    const scope_exit & ) [private], [delete]
```

Граф вызовов:

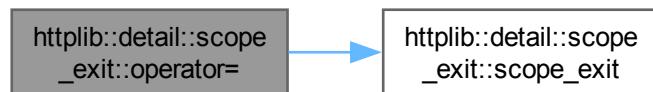


6.35.3 Методы

6.35.3.1 `operator=()` [1/2]

```
void httpplib::detail::scope_exit::operator= (
    const scope_exit & ) [private], [delete]
```

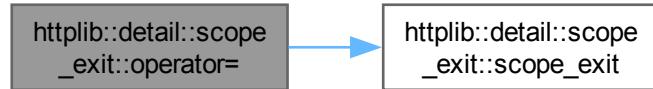
Граф вызовов:



6.35.3.2 `operator=()` [2/2]

```
scope_exit & httpplib::detail::scope_exit::operator= (
    scope_exit && ) [private], [delete]
```

Граф вызовов:



6.35.3.3 `release()`

```
void httpplib::detail::scope_exit::release () [inline]
```

См. определение в файле [httpplib.h](#) строка 424
00424 { this->`execute_on_destruction` = `false`; }

6.35.4 Данные класса

6.35.4.1 `execute_on_destruction`

```
bool httpplib::detail::scope_exit::execute_on_destruction [private]
```

См. определение в файле [httpplib.h](#) строка 432

6.35.4.2 `exit_function`

```
std::function<void(void)> httpplib::detail::scope_exit::exit_function [private]
```

См. определение в файле [httpplib.h](#) строка [431](#)

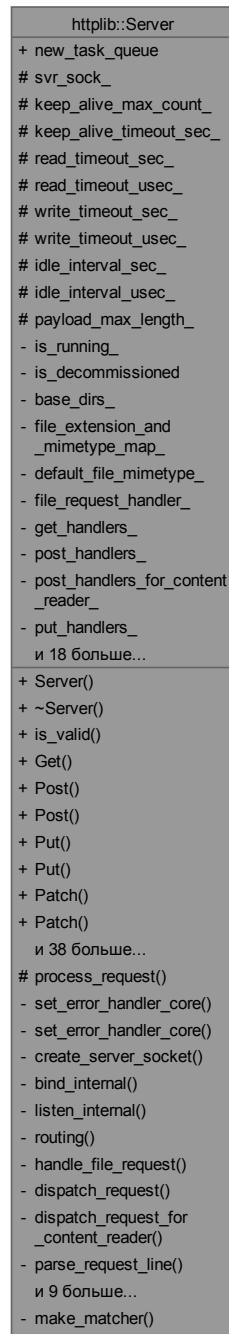
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

6.36 Класс `httpplib::Server`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::Server`:



Классы

- struct `MountPointEntry`

Открытые типы

- enum class `HandlerResponse` { `Handled` , `Unhandled` }

- using `Handler` = std::function<void(const `Request` &, `Response` &) >
- using `ExceptionHandler`
- using `HandlerWithResponse`
- using `HandlerWithContentReader`
- using `Expect100ContinueHandler`

Открытые члены

- `Server ()`
- virtual `~Server ()`
- virtual bool `is_valid () const`
- `Server & Get (const std::string &pattern, Handler handler)`
- `Server & Post (const std::string &pattern, Handler handler)`
- `Server & Post (const std::string &pattern, HandlerWithContentReader handler)`
- `Server & Put (const std::string &pattern, Handler handler)`
- `Server & Put (const std::string &pattern, HandlerWithContentReader handler)`
- `Server & Patch (const std::string &pattern, Handler handler)`
- `Server & Patch (const std::string &pattern, HandlerWithContentReader handler)`
- `Server & Delete (const std::string &pattern, Handler handler)`
- `Server & Delete (const std::string &pattern, HandlerWithContentReader handler)`
- `Server & Options (const std::string &pattern, Handler handler)`
- `bool set_base_dir (const std::string &dir, const std::string &mount_point=std::string())`
- `bool set_mount_point (const std::string &mount_point, const std::string &dir, Headers headers=Headers())`
- `bool remove_mount_point (const std::string &mount_point)`
- `Server & set_file_extension_and_mimetype_mapping (const std::string &ext, const std::string &mime)`
- `Server & set_default_file_mimetype (const std::string &mime)`
- `Server & set_file_request_handler (Handler handler)`
- template<class ErrorHandlerFunc>
 `Server & set_error_handler (ErrorHandlerFunc &&handler)`
- `Server & set_exception_handler (ExceptionHandler handler)`
- `Server & set_pre_routing_handler (HandlerWithResponse handler)`
- `Server & set_post_routing_handler (Handler handler)`
- `Server & set_expect_100_continue_handler (Expect100ContinueHandler handler)`
- `Server & set_logger (Logger logger)`
- `Server & set_address_family (int family)`
- `Server & set_tcp_nodelay (bool on)`
- `Server & set_ipv6_v6only (bool on)`
- `Server & set_socket_options (SocketOptions socket_options)`
- `Server & set_default_headers (Headers headers)`
- `Server & set_header_writer (std::function< ssize_t(Stream &, Headers &) > const &writer)`
- `Server & set_keep_alive_max_count (size_t count)`
- `Server & set_keep_alive_timeout (time_t sec)`
- `Server & set_read_timeout (time_t sec, time_t usec=0)`
- template<class Rep, class Period>
 `Server & set_read_timeout (const std::chrono::duration< Rep, Period > &duration)`
- `Server & set_write_timeout (time_t sec, time_t usec=0)`
- template<class Rep, class Period>
 `Server & set_write_timeout (const std::chrono::duration< Rep, Period > &duration)`
- `Server & set_idle_interval (time_t sec, time_t usec=0)`
- template<class Rep, class Period>
 `Server & set_idle_interval (const std::chrono::duration< Rep, Period > &duration)`
- `Server & set_payload_max_length (size_t length)`

- `bool bind_to_port (const std::string &host, int port, int socket_flags=0)`
- `int bind_to_any_port (const std::string &host, int socket_flags=0)`
- `bool listen_after_bind ()`
- `bool listen (const std::string &host, int port, int socket_flags=0)`
- `bool is_running () const`
- `void wait_until_ready () const`
- `void stop ()`
- `void decommission ()`

Открытые атрибуты

- `std::function< TaskQueue *(void)> new_task_queue`

Зашитченные члены

- `bool process_request (Stream &strm, const std::string &remote_addr, int remote_port, const std::string &local_addr, int local_port, bool close_connection, bool &connection_closed, const std::function< void(Request &)> &setup_request)`

Зашитченные данные

- `std::atomic< socket_t > svr_sock_ { (-1) }`
- `size_t keep_alive_max_count_ = 100`
- `time_t keep_alive_timeout_sec_ = 5`
- `time_t read_timeout_sec_ = 5`
- `time_t read_timeout_usec_ = 0`
- `time_t write_timeout_sec_ = 5`
- `time_t write_timeout_usec_ = 0`
- `time_t idle_interval_sec_ = 0`
- `time_t idle_interval_usec_ = 0`
- `size_t payload_max_length_ = ((std::numeric_limits<size_t>::max)())`

Закрытые типы

- `using Handlers`
- `using HandlersForContentReader`

Закрытые члены

- `Server & set_error_handler_core (HandlerWithResponse handler, std::true_type)`
- `Server & set_error_handler_core (Handler handler, std::false_type)`
- `socket_t create_server_socket (const std::string &host, int port, int socket_flags, SocketOptions socket_options) const`
- `int bind_internal (const std::string &host, int port, int socket_flags)`
- `bool listen_internal ()`
- `bool routing (Request &req, Response &res, Stream &strm)`
- `bool handle_file_request (const Request &req, Response &res, bool head=false)`
- `bool dispatch_request (Request &req, Response &res, const Handlers &handlers) const`
- `bool dispatch_request_for_content_reader (Request &req, Response &res, ContentReader content_reader, const HandlersForContentReader &handlers) const`
- `bool parse_request_line (const char *s, Request &req) const`

- void `apply_ranges` (const `Request` &req, `Response` &res, std::string &content_type, std::string &boundary) const
- bool `write_response` (`Stream` &strm, bool close_connection, `Request` &req, `Response` &res)
- bool `write_response_with_content` (`Stream` &strm, bool close_connection, const `Request` &req, `Response` &res)
- bool `write_response_core` (`Stream` &strm, bool close_connection, const `Request` &req, `Response` &res, bool need_apply_ranges)
- bool `write_content_with_provider` (`Stream` &strm, const `Request` &req, `Response` &res, const std::string &boundary, const std::string &content_type)
- bool `read_content` (`Stream` &strm, `Request` &req, `Response` &res)
- bool `read_content_with_content_receiver` (`Stream` &strm, `Request` &req, `Response` &res, `ContentReceiver` receiver, `MultipartContentHeader` multipart_header, `ContentReceiver` multipart_receive)
- bool `read_content_core` (`Stream` &strm, `Request` &req, `Response` &res, `ContentReceiver` receiver, `MultipartContentHeader` multipart_header, `ContentReceiver` multipart_receive) const
- virtual bool `process_and_close_socket` (`socket_t` sock)

Закрытые статические члены

- static std::unique_ptr< `detail::MatcherBase` > `make_matcher` (const std::string &pattern)

Закрытые данные

- std::atomic< bool > `is_running_` {false}
- std::atomic< bool > `is_decommissioned_` {false}
- std::vector< `MountPointEntry` > `base_dirs_`
- std::map< std::string, std::string > `file_extension_and_mimetype_map_`
- std::string `default_file_mimetype_` = "application/octet-stream"
- `Handler file_request_handler_`
- `Handlers get_handlers_`
- `Handlers post_handlers_`
- `HandlersForContentReader post_handlers_for_content_reader_`
- `Handlers put_handlers_`
- `HandlersForContentReader put_handlers_for_content_reader_`
- `Handlers patch_handlers_`
- `HandlersForContentReader patch_handlers_for_content_reader_`
- `Handlers delete_handlers_`
- `HandlersForContentReader delete_handlers_for_content_reader_`
- `Handlers options_handlers_`
- `HandlerWithResponse error_handler_`
- `ExceptionHandler exception_handler_`
- `HandlerWithResponse pre_routing_handler_`
- `Handler post_routing_handler_`
- `Expect100ContinueHandler expect_100_continue_handler_`
- `Logger logger_`
- int `address_family_` = AF_UNSPEC
- bool `tcp_nodelay_` = false
- bool `ipv6_v6only_` = false
- `SocketOptions socket_options_` = default_socket_options
- `Headers default_headers_`
- std::function< ssize_t(`Stream` &, `Headers` &)> `header_writer_`

6.36.1 Подробное описание

См. определение в файле `httpplib.h` строка [948](#)

6.36.2 Определения типов

6.36.2.1 ExceptionHandler

```
using httpplib::Server::ExceptionHandler
```

Инициализатор

```
std::function<void(const Request &, Response &, std::exception_ptr ep)>
```

См. определение в файле `httpplib.h` строка [952](#)

6.36.2.2 Expect100ContinueHandler

```
using httpplib::Server::Expect100ContinueHandler
```

Инициализатор

```
std::function<int(const Request &, Response &)>
```

См. определение в файле `httpplib.h` строка [965](#)

6.36.2.3 Handler

```
using httpplib::Server::Handler = std::function<void(const Request &, Response &)>
```

См. определение в файле `httpplib.h` строка [950](#)

6.36.2.4 Handlers

```
using httpplib::Server::Handlers [private]
```

Инициализатор

```
std::vector<std::pair<std::unique_ptr<detail::MatcherBase>, Handler>
```

См. определение в файле `httpplib.h` строка [1067](#)

6.36.2.5 HandlersForContentReader

```
using httpplib::Server::HandlersForContentReader [private]
```

Инициализатор

```
std::vector<std::pair<std::unique_ptr<detail::MatcherBase>, HandlerWithContentReader>
```

См. определение в файле `httpplib.h` строка [1069](#)

6.36.2.6 HandlerWithContentReader

```
using httpplib::Server::HandlerWithContentReader
```

Инициализатор

```
std::function<void(  
    const Request &, Response &, const ContentReader &content_reader)>
```

См. определение в файле `httpplib.h` строка 962

6.36.2.7 HandlerWithResponse

```
using httpplib::Server::HandlerWithResponse
```

Инициализатор

```
std::function<HandlerResponse(const Request &, Response &)>
```

См. определение в файле `httpplib.h` строка 959

6.36.3 Перечисления

6.36.3.1 HandlerResponse

```
enum class httpplib::Server::HandlerResponse [strong]
```

Элементы перечислений

Handled	
Unhandled	

См. определение в файле `httpplib.h` строка 955

```
00955           {  
00956     Handled,  
00957     Unhandled,  
00958   };
```

6.36.4 Конструктор(ы)

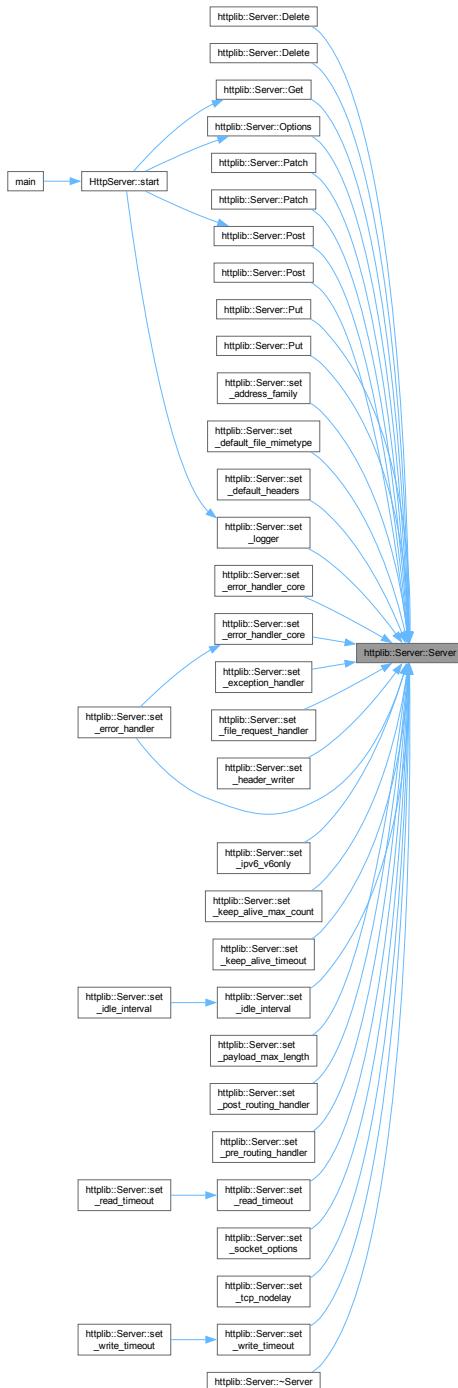
6.36.4.1 Server()

```
httpplib::Server::Server () [inline]
```

См. определение в файле `httpplib.h` строка 6309

```
06310   : new task_queue(  
06311     [] { return new ThreadPool(CPPHTTPPLIB_THREAD_POOL_COUNT); }) {  
06312 #ifndef WIN32  
06313   signal(SIGPIPE, SIG_IGN);  
06314 #endif  
06315 }
```

Граф вызова функций:



6.36.4.2 ~Server()

`httpplib::Server::~Server () [inline], [virtual], [default]`

Граф вызовов:



6.36.5 Методы

6.36.5.1 apply_ranges()

```

void http://Server::apply_ranges (
    const Request & req,
    Response & res,
    std::string & content_type,
    std::string & boundary) const [inline], [private]
    
```

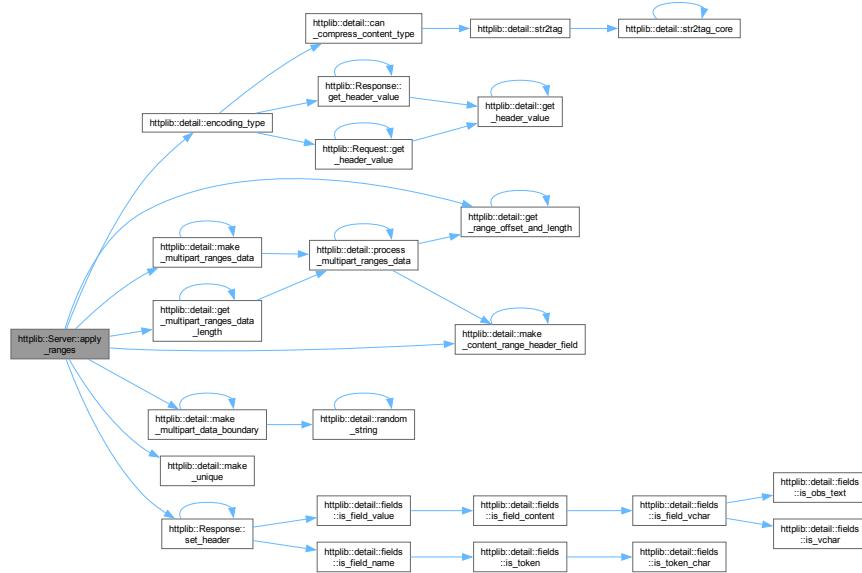
См. определение в файле [http://Server.h](#) строка 7124

```

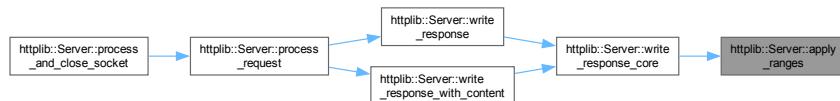
07126     {
07127     if (req.ranges.size() > 1 && res.status == StatusCode::PartialContent_206) {
07128         auto it = res.headers.find("Content-Type");
07129         if (it != res.headers.end()) {
07130             content_type = it->second;
07131             res.headers.erase(it);
07132         }
07133         boundary = detail::make_multipart_data_boundary();
07135         res.set_header("Content-Type",
07136             "multipart/byteranges; boundary=" + boundary);
07138     }
07139     auto type = detail::encoding_type(req, res);
07141
07142     if (res.body.empty()) {
07143         if (res.content_length_ > 0) {
07144             size_t length = 0;
07145             if (req.ranges.empty() || res.status != StatusCode::PartialContent_206) {
07146                 length = res.content_length_;
07147             } else if (req.ranges.size() == 1) {
07148                 auto offset_and_length = detail::get_range_offset_and_length(
07149                     req.ranges[0], res.content_length_);
07150                 length = offset_and_length.second;
07152             }
07153             auto content_range = detail::make_content_range_header_field(
07154                 offset_and_length, res.content_length_);
07155             res.set_header("Content-Range", content_range);
07156         } else {
07157             length = detail::get_multipart_ranges_data_length(
07158                 req, boundary, content_type, res.content_length_);
07159         }
07160         res.set_header("Content-Length", std::to_string(length));
07161     } else {
07162         if (res.content_provider_) {
07163             if (res.is_chunked_content_provider_) {
07164                 res.set_header("Transfer-Encoding", "chunked");
07165                 if (type == detail::EncodingType::Gzip) {
07166                     res.set_header("Content-Encoding", "gzip");
07167                 } else if (type == detail::EncodingType::Brotli) {
07168                     res.set_header("Content-Encoding", "br");
07169                 } else if (type == detail::EncodingType::Zstd) {
07170                     res.set_header("Content-Encoding", "zstd");
07171                 }
07172             }
07173         }
07174     }
    
```

```
07175 } else {
07176     if (req.ranges.empty() || res.status != StatusCode::PartialContent_206) {
07177     ;
07178 } else if (req.ranges.size() == 1) {
07179     auto offset_and_length =
07180         detail::get_range_offset_and_length(req.ranges[0], res.body.size());
07181     auto offset = offset_and_length.first;
07182     auto length = offset_and_length.second;
07183
07184     auto content_range = detail::make_content_range_header_field(
07185         offset_and_length, res.body.size());
07186     res.set_header("Content-Range", content_range);
07187
07188     assert(offset + length <= res.body.size());
07189     res.body = res.body.substr(offset, length);
07190 } else {
07191     std::string data;
07192     detail::make_multipart_ranges_data(req, res, boundary, content_type,
07193                                         res.body.size(), data);
07194     res.body.swap(data);
07195 }
07196
07197 if (type != detail::EncodingType::None) {
07198     std::unique_ptr<detail::compressor> compressor;
07199     std::string content_encoding;
07200
07201     if (type == detail::EncodingType::Gzip) {
07202 #ifndef CPPHTTPLIB_ZLIB_SUPPORT
07203         compressor = detail::make_unique<detail::gzip_compressor>();
07204         content_encoding = "gzip";
07205 #endif
07206     } else if (type == detail::EncodingType::Brotli) {
07207 #ifndef CPPHTTPLIB_BROTLI_SUPPORT
07208         compressor = detail::make_unique<detail::brotli_compressor>();
07209         content_encoding = "br";
07210 #endif
07211     } else if (type == detail::EncodingType::Zstd) {
07212 #ifndef CPPHTTPLIB_ZSTD_SUPPORT
07213         compressor = detail::make_unique<detail::zstd_compressor>();
07214         content_encoding = "zstd";
07215 #endif
07216     }
07217
07218     if (compressor) {
07219         std::string compressed;
07220         if (compressor->compress(res.body.data(), res.body.size(), true,
07221             [&](const char *data, size_t data_len) {
07222                 compressed.append(data, data_len);
07223                 return true;
07224             })) {
07225             res.body.swap(compressed);
07226             res.set_header("Content-Encoding", content_encoding);
07227         }
07228     }
07229 }
07230
07231 auto length = std::to_string(res.body.size());
07232 res.set_header("Content-Length", length);
07233 }
07234 }
```

Граф вызовов:



Граф вызова функций:



6.36.5.2 bind_internal()

```

int http://Server::bind_internal (
    const std::string & host,
    int port,
    int socket_flags) [inline], [private]
    
```

См. определение в файле [http://lib.h](#) строка 6934

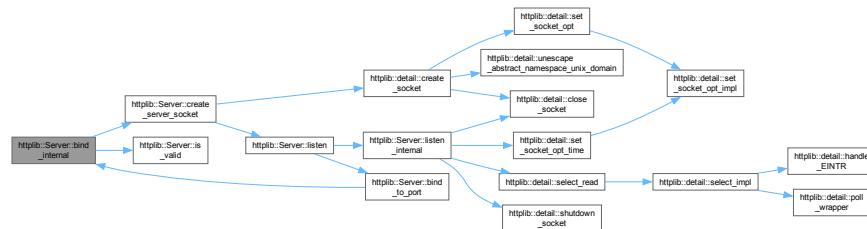
```

06935     {
06936     if (is_decommissioned) { return -1; }
06937
06938     if (!is_valid()) { return -1; }
06939
06940     svr_sock_ = create_server_socket(host, port, socket_flags, socket_options_);
06941     if (svr_sock_ == INVALID_SOCKET) { return -1; }
06942
06943     if (port == 0) {
06944         struct sockaddr_storage addr;
06945         socklen_t addr_len = sizeof(addr);
06946         if (getsockname(svr_sock_, reinterpret_cast<struct sockaddr*>(&addr),
06947                         &addr_len) == -1) {
06948             return -1;
06949         }
06950         if (addr.ss_family == AF_INET) {
06951             return ntohs(reinterpret_cast<struct sockaddr_in*>(&addr)->sin_port);
06952         } else if (addr.ss_family == AF_INET6) {
06953             return ntohs(reinterpret_cast<struct sockaddr_in6*>(&addr)->sin6_port);
06954         }
06955     }
06956
06957     if (bind(svr_sock_, (const struct sockaddr*)&addr, sizeof(addr)) == -1) {
06958         return -1;
06959     }
06960
06961     if (listen(svr_sock_, backlog) == -1) {
06962         return -1;
06963     }
06964
06965     return 0;
06966 }
    
```

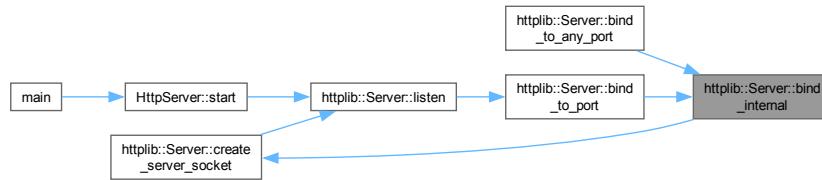
```

06954     } else {
06955         return -1;
06956     }
06957 } else {
06958     return port;
06959 }
06960 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.3 bind_to_any_port()

```

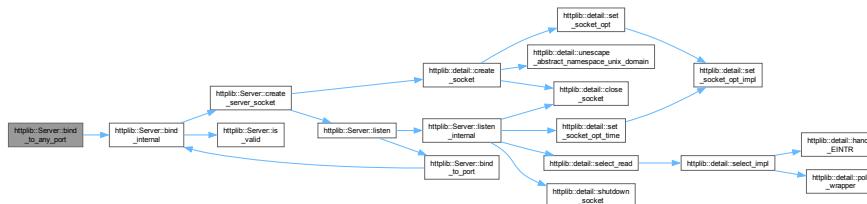
int httpplib::Server::bind_to_any_port (
    const std::string & host,
    int socket_flags = 0) [inline]
```

См. определение в файле [httpplib.h](#) строка 6542

```

06542
06543     auto ret = bind_internal(host, 0, socket_flags);
06544     if (ret == -1) { is_decommissioned = true; }
06545     return ret;
06546 }
```

Граф вызовов:



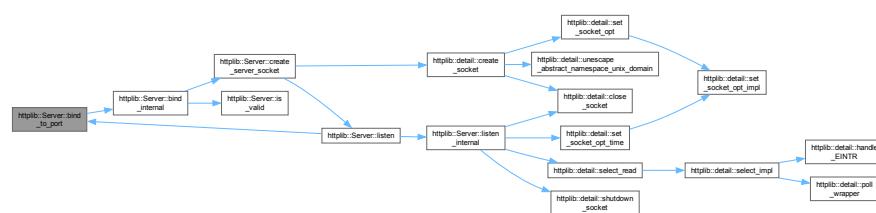
6.36.5.4 `bind_to_port()`

```
bool httpplib::Server::bind_to_port (
    const std::string & host,
    int port,
    int socket_flags = 0) [inline]
```

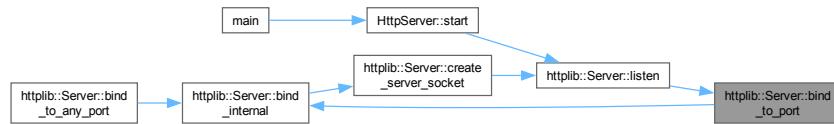
См. определение в файле `httpplib.h` строка 6536

```
06537     {
06538     auto ret = bind_internal(host, port, socket_flags);
06539     if (ret == -1) { is_decommissioned = true; }
06540     return ret >= 0;
06541 }
```

Граф вызовов:



Граф вызова функции:

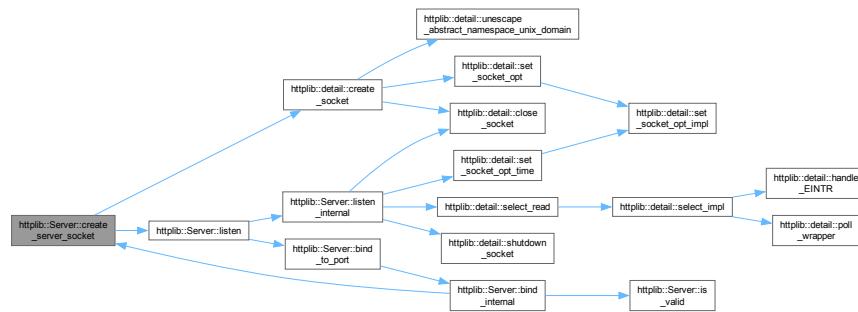
6.36.5.5 `create_server_socket()`

```
socket_t httpplib::Server::create_server_socket (
    const std::string & host,
    int port,
    int socket_flags,
    SocketOptions socket_options) const [inline], [private]
```

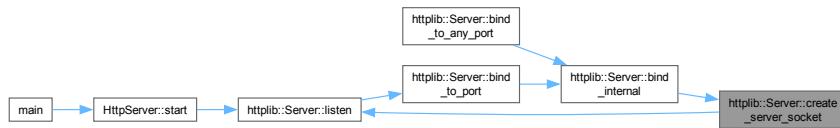
См. определение в файле `httpplib.h` строка 6919

```
06921     {
06922     return detail::create_socket(
06923         host, std::string(), port, address_family_, socket_flags, tcp_nodelay_,
06924         ipv6_v6only_, std::move(socket_options),
06925         [(socket_t sock, struct sockaddr_in& ai, bool & /*quit*/) -> bool {
06926             if (::bind(sock, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen))) {
06927                 return false;
06928             }
06929             if (::listen(sock, CPPHTTPPLIB_LISTEN_BACKLOG)) { return false; }
06930             return true;
06931         });
06932 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.6 decommission()

`void http://Server::decommission () [inline]`

См. определение в файле [http://Server.h](#) строка 6573

06573 { `is_decommissioned = true;` }

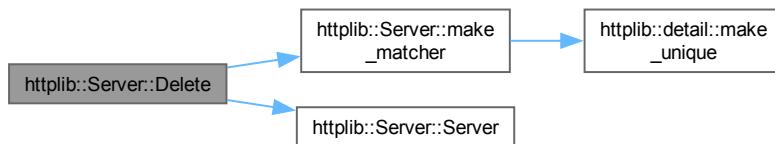
6.36.5.7 Delete() [1/2]

```
Server & http://Server::Delete (
    const std::string & pattern,
    Handler handler) [inline]
```

См. определение в файле [http://Server.h](#) строка 6369

```
06369 {
06370     delete_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06371     return *this;
06372 }
```

Граф вызовов:



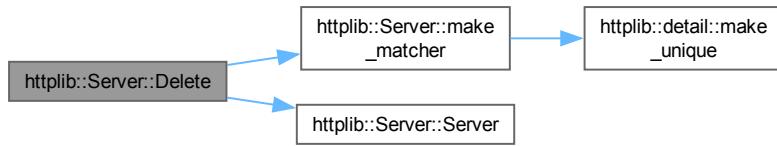
6.36.5.8 `Delete()` [2/2]

```
Server & httpplib::Server::Delete (
    const std::string & pattern,
    HandlerWithContentReader handler) [inline]
```

См. определение в файле `httpplib.h` строка 6374

```
06375     {
06376     delete_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06377                                         std::move(handler));
06378     return *this;
06379 }
```

Граф вызовов:

6.36.5.9 `dispatch_request()`

```
bool httpplib::Server::dispatch_request (
    Request & req,
    Response & res,
    const Handlers & handlers) const [inline], [private]
```

См. определение в файле `httpplib.h` строка 7110

```
07111     {
07112     for (const auto &x : handlers) {
07113         const auto &zmatcher = x.first;
07114         const auto &handler = x.second;
07115
07116         if (matcher->match(req)) {
07117             handler(req, res);
07118             return true;
07119         }
07120     }
07121     return false;
07122 }
```

Граф вызова функции:



6.36.5.10 `dispatch_request_for_content_reader()`

```
bool httpplib::Server::dispatch_request_for_content_reader (
    Request & req,
    Response & res,
    ContentReader content_reader,
    const HandlersForContentReader & handlers) const [inline], [private]
```

См. определение в файле `httpplib.h` строка [7236](#)

```
07238     {
07239         for (const auto &x : handlers) {
07240             const auto &matcher = x.first;
07241             const auto &handler = x.second;
07242
07243             if (matcher->match(req)) {
07244                 handler(req, res, content_reader);
07245                 return true;
07246             }
07247         }
07248     return false;
07249 }
```

Граф вызова функции:

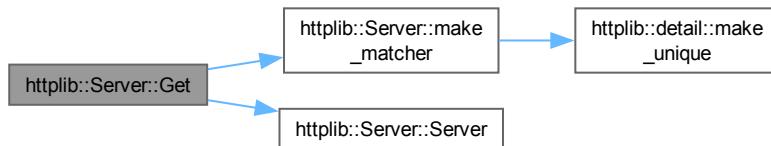
6.36.5.11 `Get()`

```
Server & httpplib::Server::Get (
    const std::string & pattern,
    Handler handler) [inline]
```

См. определение в файле `httpplib.h` строка [6328](#)

```
06328     {
06329         get_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06330         return *this;
06331     }
```

Граф вызовов:



Граф вызова функции:



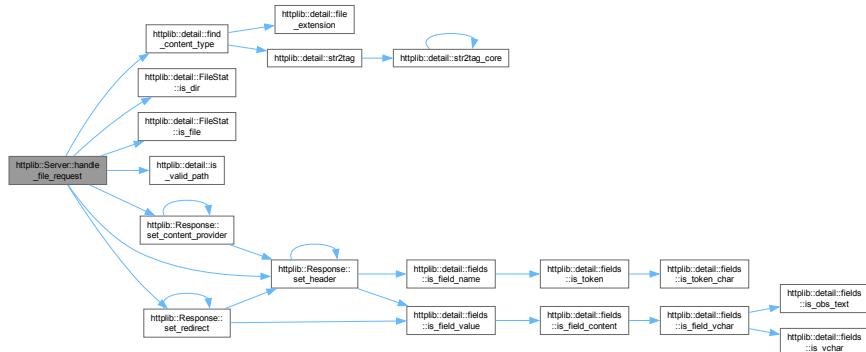
6.36.5.12 `handle_file_request()`

```
bool httpplib::Server::handle_file_request (
    const Request & req,
    Response & res,
    bool head = false) [inline], [private]
```

См. определение в файле `httpplib.h` строка 6872

```
06873
06874     for (const auto &zentry : base_dirs_) {
06875         // Prefix match
06876         if (!req.path.compare(0, entry.mount_point.size(), entry.mount_point)) {
06877             std::string sub_path = "/" + req.path.substr(entry.mount_point.size());
06878             if (detail::is_valid_path(sub_path)) {
06879                 auto path = entry.base_dir + sub_path;
06880                 if (path.back() == '/') { path += "index.html"; }
06881
06882                 detail::FileStat stat(path);
06883
06884                 if (stat.is_dir()) {
06885                     res.set_redirect(sub_path + "/", StatusCode::MovedPermanently_301);
06886                     return true;
06887                 }
06888
06889                 if (stat.is_file()) {
06890                     for (const auto &kv : entry.headers) {
06891                         res.set_header(kv.first, kv.second);
06892                     }
06893
06894                     auto mm = std::make_shared<detail::mmap>(path.c_str());
06895                     if (!mm->is_open()) { return false; }
06896
06897                     res.set_content_provider(
06898                         mm->size(),
06899                         detail::find_content_type(path, file_extension_and_mimetype_map_,
06900                                         default_file_mimetype),
06901                         [mm](size_t offset, size_t length, DataSink & sink) -> bool {
06902                             sink.write(mm->data() + offset, length);
06903                             return true;
06904                         });
06905
06906                     if (!head && file_request_handler_) {
06907                         file_request_handler_(req, res);
06908                     }
06909
06910                     return true;
06911                 }
06912             }
06913         }
06914     }
06915     return false;
06916 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.13 is_running()

`bool httplib::Server::is_running () const [inline]`

См. определение в файле [httplib.h](#) строка 6555

```
06555 { return is_running_; }
```

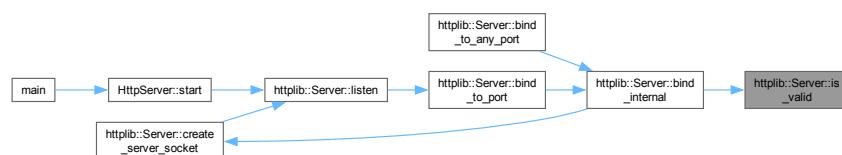
6.36.5.14 is_valid()

`bool httplib::Server::is_valid () const [inline], [virtual]`

См. определение в файле [httplib.h](#) строка 7420

```
07420 { return true; }
```

Граф вызова функции:



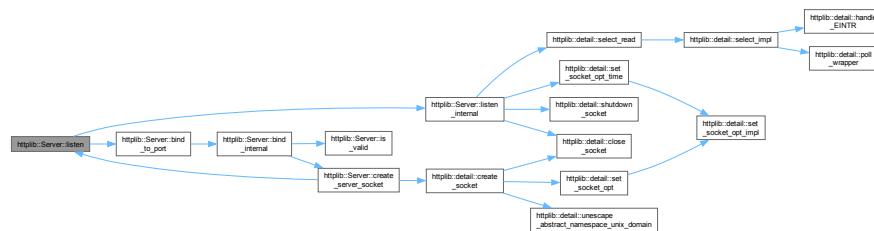
6.36.5.15 `listen()`

```
bool httpplib::Server::listen (
    const std::string & host,
    int port,
    int socket_flags = 0) [inline]
```

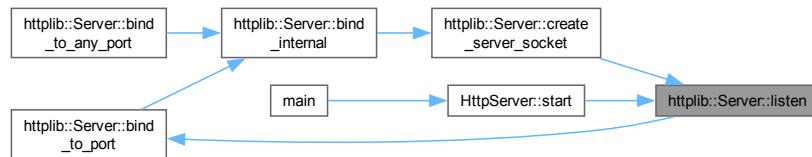
См. определение в файле `httpplib.h` строка 6550

```
06551     {
06552     return bind_to_port(host, port, socket_flags) && listen_internal();
06553 }
```

Граф вызовов:



Граф вызова функции:

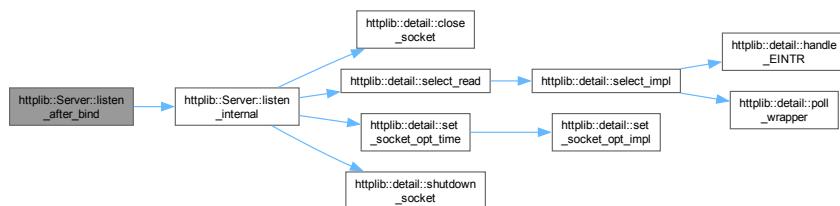
6.36.5.16 `listen_after_bind()`

```
bool httpplib::Server::listen_after_bind () [inline]
```

См. определение в файле `httpplib.h` строка 6548

```
06548 { return listen_internal(); }
```

Граф вызовов:



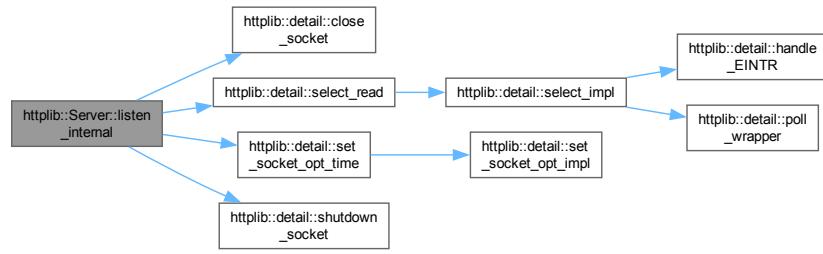
6.36.5.17 `listen_internal()`

```
bool httpplib::Server::listen_internal () [inline], [private]
```

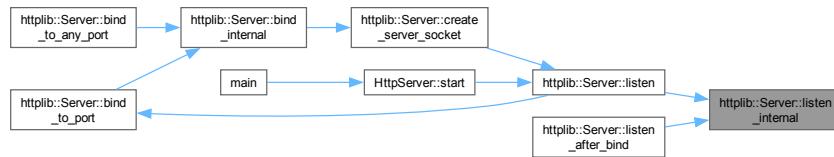
См. определение в файле [httpplib.h](#) строка 6962

```
06962     {
06963     if (is_decommissioned) { return false; }
06964     auto ret = true;
06965     is_running_ = true;
06966     auto se = detail::scope_exit([&]() { is_running_ = false; });
06967     {
06968         std::unique_ptr<TaskQueue> task_queue(new_task_queue());
06969         while (svr_sock_ != INVALID_SOCKET) {
06970             #ifndef _WIN32
06971                 if (idle_interval_sec_ > 0 || idle_interval_usec_ > 0) {
06972                     #endif
06973                     auto val = detail::select_read(svr_sock_, idle_interval_sec_,
06974                                         idle_interval_usec_);
06975                     if (val == 0) { // Timeout
06976                         task_queue->on_idle();
06977                         continue;
06978                     }
06979 #ifndef _WIN32
06980                 }
06981             #endif
06982             #if defined _WIN32
06983                 // sockets connected via WASAAccept inherit flags NO_HANDLE_INHERIT,
06984                 // OVERLAPPED
06985                 socket_t sock = WSAAccept(svr_sock_, nullptr, nullptr, nullptr, 0);
06986             #elif defined SOCK_CLOEXEC
06987                 socket_t sock = accept4(svr_sock_, nullptr, nullptr, SOCK_CLOEXEC);
06988             #else
06989                 socket_t sock = accept(svr_sock_, nullptr, nullptr);
06990             #endif
06991             if (sock == INVALID_SOCKET) {
06992                 if (errno == EMFILE) {
06993                     // The per-process limit of open file descriptors has been reached.
06994                     // Try to accept new connections after a short sleep.
06995                     std::this_thread::sleep_for(std::chrono::microseconds{1});
06996                     continue;
06997                 } else if (errno == EINTR || errno == EAGAIN) {
06998                     continue;
06999                 }
07000                 if (svr_sock_ != INVALID_SOCKET) {
07001                     detail::close_socket(svr_sock_);
07002                     ret = false;
07003                 } else {
07004                     ; // The server socket was closed by user.
07005                 }
07006                 break;
07007             }
07008             detail::set_socket_opt_time(sock, SOL_SOCKET, SO_RCVTIMEO,
07009                                         read_timeout_sec_, read_timeout_usec_);
07010             detail::set_socket_opt_time(sock, SOL_SOCKET, SO_SNDDTIMEO,
07011                                         write_timeout_sec_, write_timeout_usec_);
07012         }
07013         if (!task_queue->enqueue(
07014             [this, sock]() { process_and_close_socket(sock); })) {
07015             detail::shutdown_socket(sock);
07016             detail::close_socket(sock);
07017         }
07018     }
07019     task_queue->shutdown();
07020 }
07021 is_decommissioned = !ret;
07022 return ret;
07023 }
```

Граф вызовов:



Граф вызова функции:



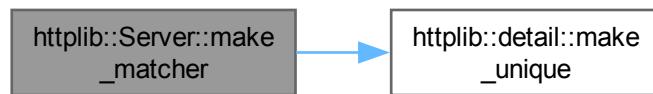
6.36.5.18 make_matcher()

```
std::unique_ptr< detail::MatcherBase > httplib::Server::make_matcher (
    const std::string & pattern) [inline], [static], [private]
```

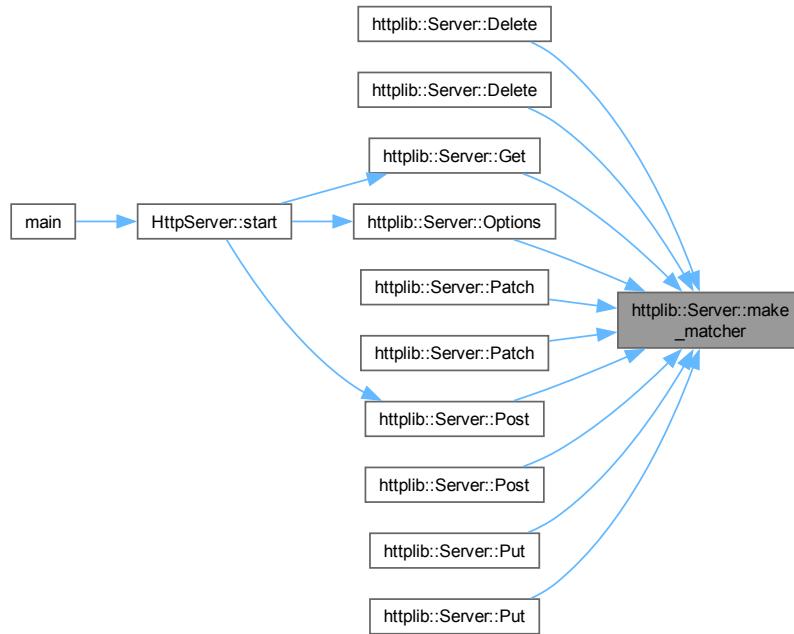
См. определение в файле [httplib.h](#) строка 6320

```
06320   if (pattern.find("/") != std::string::npos) {
06321     return detail::make_unique<detail::PathParamsMatcher>(pattern);
06322   } else {
06323     return detail::make_unique<detail::RegexMatcher>(pattern);
06324   }
06325 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.19 Options()

```

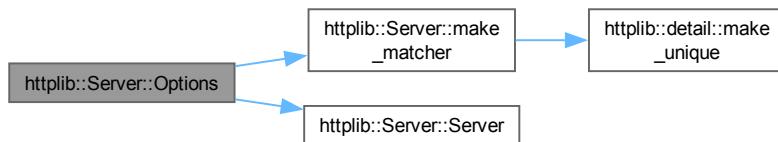
Server & httpplib::Server::Options (
    const std::string & pattern,
    Handler handler) [inline]
  
```

См. определение в файле `httpplib.h` строка 6381

```

06381 {
06382     options_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06383     return *this;
06384 }
  
```

Граф вызовов:



Граф вызова функции:



6.36.5.20 `parse_request_line()`

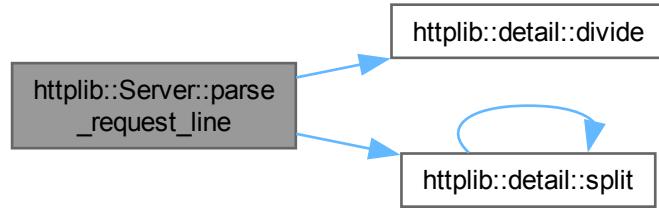
```
bool httpplib::Server::parse_request_line (
    const char * s,
    Request & req) const [inline], [private]
```

См. определение в файле [httpplib.h](#) строка 6575

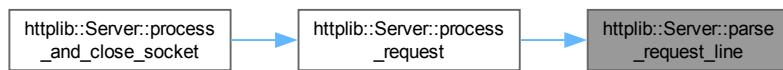
```

06575                                     {
06576     auto len = strlen(s);
06577     if (len < 2 || s[len - 2] != '\r' || s[len - 1] != '\n') { return false; }
06578     len -= 2;
06579
06580     {
06581         size_t count = 0;
06582
06583         detail::split(s + len, ' ', [&](const char *b, const char *e) {
06584             switch (count) {
06585                 case 0: req.method = std::string(b, e); break;
06586                 case 1: req.target = std::string(b, e); break;
06587                 case 2: req.version = std::string(b, e); break;
06588                 default: break;
06589             }
06590             count++;
06591         });
06592
06593         if (count != 3) { return false; }
06594     }
06595
06596     thread_local const std::set<std::string> methods{
06597         "GET", "HEAD", "POST", "PUT", "DELETE",
06598         "CONNECT", "OPTIONS", "TRACE", "PATCH", "PRI"};
06599
06600     if (methods.find(req.method) == methods.end()) { return false; }
06601
06602     if (req.version != "HTTP/1.1" && req.version != "HTTP/1.0") { return false; }
06603
06604     {
06605         // Skip URL fragment
06606         for (size_t i = 0; i < req.target.size(); i++) {
06607             if (req.target[i] == '#') {
06608                 req.target.erase(i);
06609                 break;
06610             }
06611         }
06612
06613         detail::divide(req.target, '?',
06614             [&](const char *lhs_data, std::size_t lhs_size,
06615                 const char *rhs_data, std::size_t rhs_size) {
06616                 req.path = detail::decode_url(
06617                     std::string(lhs_data, lhs_size), false);
06618                 detail::parse_query_text(rhs_data, rhs_size, req.params);
06619             });
06620     }
06621
06622     return true;
06623 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.21 Patch() [1/2]

```

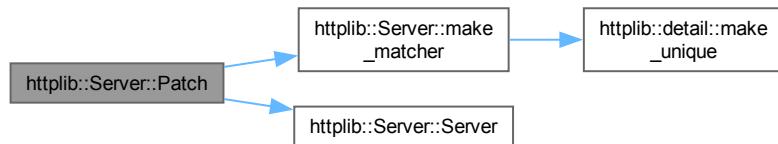
Server & httplib::Server::Patch (
    const std::string & pattern,
    Handler handler) [inline]
  
```

См. определение в файле [httplib.h](#) строка 6357

```

06357
06358     patch_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06359     return *this;
06360 }
  
```

Граф вызовов:



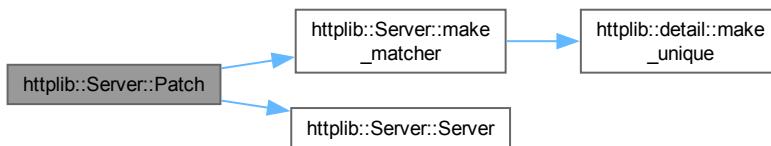
6.36.5.22 `Patch()` [2/2]

```
Server & httpplib::Server::Patch (
    const std::string & pattern,
    HandlerWithContentReader handler) [inline]
```

См. определение в файле `httpplib.h` строка 6362

```
06363     {
06364     patch_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06365                     std::move(handler));
06366     return *this;
06367 }
```

Граф вызовов:

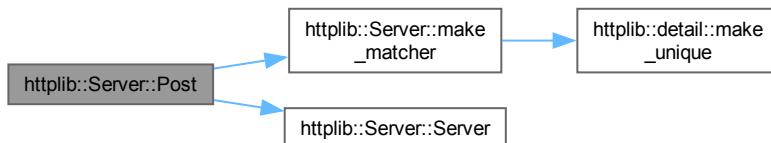
6.36.5.23 `Post()` [1/2]

```
Server & httpplib::Server::Post (
    const std::string & pattern,
    Handler handler) [inline]
```

См. определение в файле `httpplib.h` строка 6333

```
06333     {
06334     post_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06335     return *this;
06336 }
```

Граф вызовов:



Граф вызова функции:



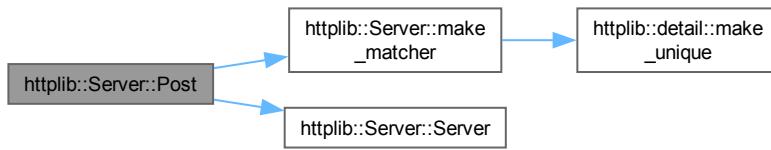
6.36.5.24 `Post()` [2/2]

```
Server & httpplib::Server::Post (
    const std::string & pattern,
    HandlerWithContentReader handler) [inline]
```

См. определение в файле `httpplib.h` строка 6338

```
06339     {
06340     post_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06341                                     std::move(handler));
06342     return *this;
06343 }
```

Граф вызовов:

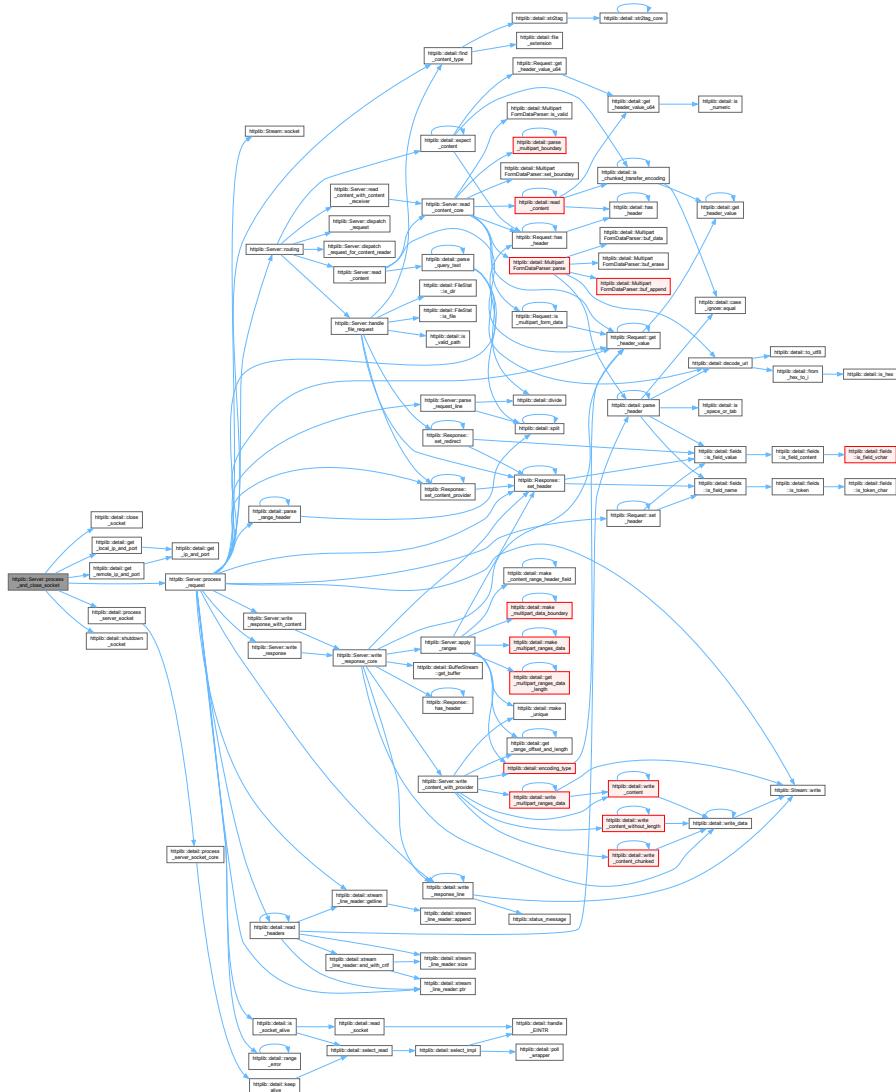
6.36.5.25 `process_and_close_socket()`

```
bool httpplib::Server::process_and_close_socket (
    socket_t sock) [inline], [private], [virtual]
```

См. определение в файле `httpplib.h` строка 7422

```
07422 {
07423     std::string remote_addr;
07424     int remote_port = 0;
07425     detail::get_remote_ip_and_port(sock, remote_addr, remote_port);
07426
07427     std::string local_addr;
07428     int local_port = 0;
07429     detail::get_local_ip_and_port(sock, local_addr, local_port);
07430
07431     auto ret = detail::process_server_socket(
07432         svr_sock_, sock, keep_alive_max_count_, keep_alive_timeout_sec_,
07433         read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
07434         write_timeout_usec_,
07435         [&](Stream &strm, bool close_connection, bool &connection_closed) {
07436             return process_request(strm, remote_addr, remote_port, local_addr,
07437                                     local_port, close_connection, connection_closed,
07438                                     nullptr);
07439         });
07440
07441     detail::shutdown_socket(sock);
07442     detail::close_socket(sock);
07443     return ret;
07444 }
```

Граф вызовов:



6.36.5.26 process_request()

```
bool httpplib::Server::process_request (
    Stream & strm,
    const std::string & remote_addr,
    int remote_port,
    const std::string & local_addr,
    int local_port,
    bool close_connection,
    bool & connection_closed,
    const std::function< void(Request &) > & setup_request) [inline], [protected]
```

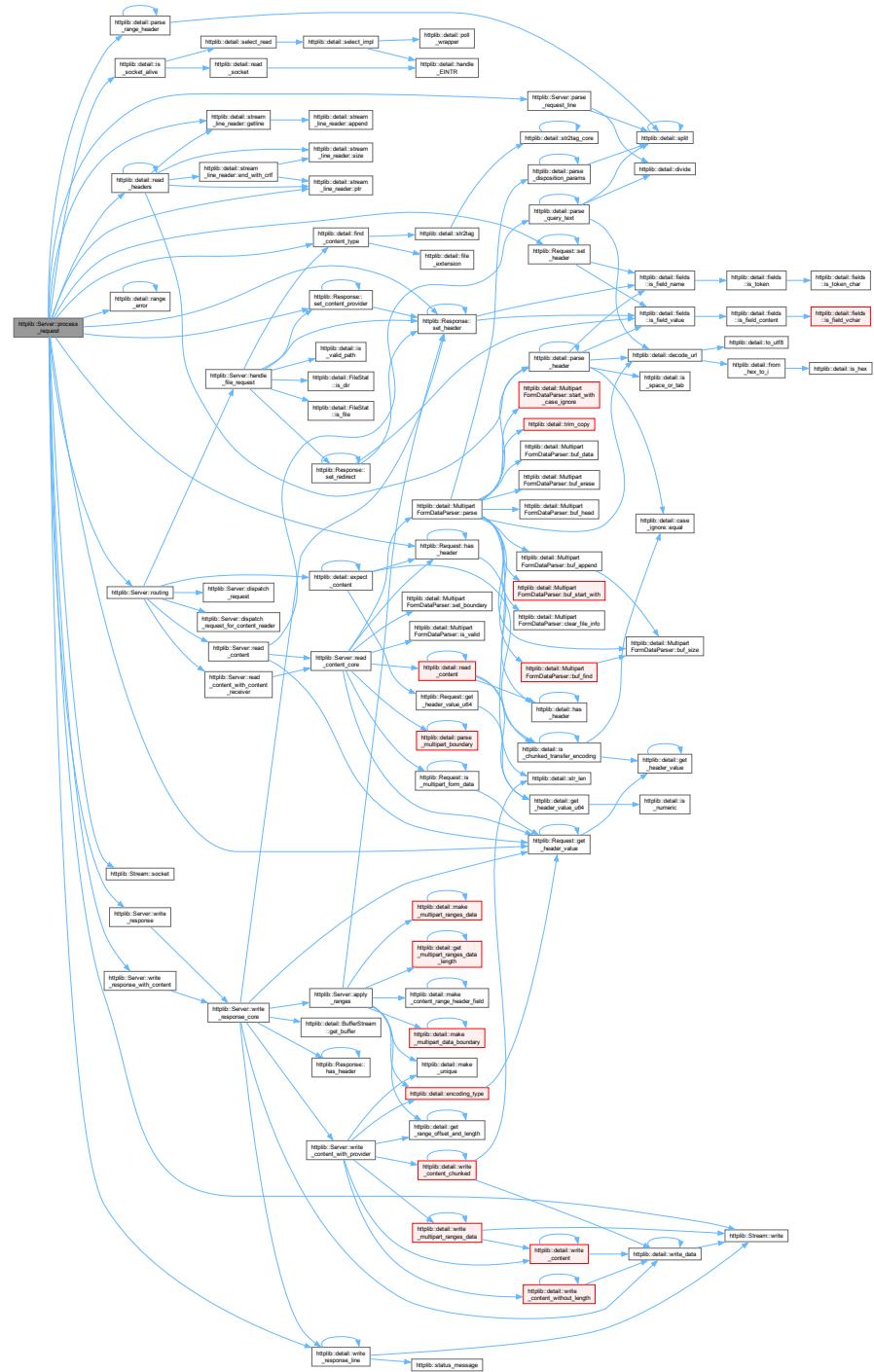
См. определение в файле `httpplib.h` строка 7252

```
07256
07257     std::array<char, 2048> buf{};
07258
07259     detail::stream_line_reader line_reader(strm, buf.data(), buf.size());
07260
```

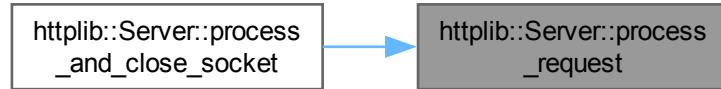
```
07261 // Connection has been closed on client
07262 if (!line_reader.getline()) { return false; }
07263
07264 Request req;
07265
07266 Response res;
07267 res.version = "HTTP/1.1";
07268 res.headers = default_headers_;
07269
07270 // Request line and headers
07271 if (!parse_request_line(line_reader.ptr(), req) ||
07272     !detail::read_headers(strm, req.headers)) {
07273     res.status = StatusCode::BadRequest_400;
07274     return write_response(strm, close_connection, req, res);
07275 }
07276
07277 // Check if the request URI doesn't exceed the limit
07278 if (req.target.size() > CPPHTTPPLIB_REQUEST_URI_MAX_LENGTH) {
07279     Headers dummy;
07280     detail::read_headers(strm, dummy);
07281     res.status = StatusCode::UriTooLong_414;
07282     return write_response(strm, close_connection, req, res);
07283 }
07284
07285 if (req.get_header_value("Connection") == "close") {
07286     connection_closed = true;
07287 }
07288
07289 if (req.version == "HTTP/1.0" &&
07290     req.get_header_value("Connection") != "Keep-Alive") {
07291     connection_closed = true;
07292 }
07293
07294 req.remote_addr = remote_addr;
07295 req.remote_port = remote_port;
07296 req.set_header("REMOTE_ADDR", req.remote_addr);
07297 req.set_header("REMOTE_PORT", std::to_string(req.remote_port));
07298
07299 req.local_addr = local_addr;
07300 req.local_port = local_port;
07301 req.set_header("LOCAL_ADDR", req.local_addr);
07302 req.set_header("LOCAL_PORT", std::to_string(req.local_port));
07303
07304 if (req.has_header("Range")) {
07305     const auto &range_header_value = req.get_header_value("Range");
07306     if (!detail::parse_range_header(range_header_value, req.ranges)) {
07307         res.status = StatusCode::RangeNotSatisfiable_416;
07308         return write_response(strm, close_connection, req, res);
07309     }
07310 }
07311
07312 if (setup_request) { setup_request(req); }
07313
07314 if (req.get_header_value("Expect") == "100-continue") {
07315     int status = StatusCode::Continue_100;
07316     if (expect_100_continue_handler_) {
07317         status = expect_100_continue_handler_(req, res);
07318     }
07319     switch (status) {
07320         case StatusCode::Continue_100:
07321         case StatusCode::ExpectationFailed_417:
07322             detail::write_response_line(strm, status);
07323             strm.write("\r\n");
07324             break;
07325         default:
07326             connection_closed = true;
07327             return write_response(strm, true, req, res);
07328     }
07329 }
07330
07331 // Setup `is_connection_closed` method
07332 req.is_connection_closed = [&]() {
07333     return !detail::is_socket_alive(strm.socket());
07334 };
07335
07336 // Routing
07337 auto routed = false;
07338 #ifndef CPPHTTPPLIB_NO_EXCEPTIONS
07339     routed = routing(req, res, strm);
07340 #else
07341     try {
07342         routed = routing(req, res, strm);
07343     } catch (std::exception &e) {
07344         if (exception_handler_) {
07345             auto ep = std::current_exception();
07346             exception_handler_(req, res, ep);
07347             routed = true;
07348         }
07349     }
07350 }
07351
07352 if (routed) {
07353     if (expect_100_continue_handler_) {
07354         detail::write_response_line(strm, StatusCode::Continue_100);
07355         strm.write("\r\n");
07356     }
07357     if (exception_handler_) {
07358         exception_handler_(req, res, ep);
07359     }
07360 }
07361
07362 if (connection_closed) {
07363     if (expect_100_continue_handler_) {
07364         detail::write_response_line(strm, StatusCode::ExpectationFailed_417);
07365         strm.write("\r\n");
07366     }
07367     if (exception_handler_) {
07368         exception_handler_(req, res, ep);
07369     }
07370 }
```

```
07348 } else {
07349     res.status = StatusCode::InternalServerError_500;
07350     std::string val;
07351     auto s = e.what();
07352     for (size_t i = 0; s[i]; i++) {
07353         switch (s[i]) {
07354             case '\r': val += "\\r"; break;
07355             case '\n': val += "\\n"; break;
07356             default: val += s[i]; break;
07357         }
07358     }
07359     res.set_header("EXCEPTION_WHAT", val);
07360 }
07361 } catch (...) {
07362     if (exception_handler_) {
07363         auto ep = std::current_exception();
07364         exception_handler_(req, res, ep);
07365         routed = true;
07366     } else {
07367         res.status = StatusCode::InternalServerError_500;
07368         res.set_header("EXCEPTION_WHAT", "UNKNOWN");
07369     }
07370 }
07371 #endif
07372 if (routed) {
07373     if (res.status == -1) {
07374         res.status = req.ranges.empty() ? StatusCode::OK_200
07375                               : StatusCode::PartialContent_206;
07376     }
07377
07378 // Serve file content by using a content provider
07379 if (!res.file_content_path_.empty()) {
07380     const auto &path = res.file_content_path_;
07381     auto mm = std::make_shared<detail::mmap>(path.c_str());
07382     if (!mm->is_open()) {
07383         res.body.clear();
07384         res.content_length_ = 0;
07385         res.content_provider_ = nullptr;
07386         res.status = StatusCode::NotFound_404;
07387         return write_response(strm, close_connection, req, res);
07388     }
07389
07390     auto content_type = res.file_content_content_type_;
07391     if (content_type.empty()) {
07392         content_type = detail::find_content_type(
07393             path, file_extension_and_mimetype_map_, default_file_mimetype_);
07394     }
07395
07396     res.set_content_provider(
07397         mm->size(), content_type,
07398         [mm](size_t offset, size_t length, DataSink &sink) -> bool {
07399             sink.write(mm->data() + offset, length);
07400             return true;
07401         });
07402     }
07403
07404     if (detail::range_error(req, res)) {
07405         res.body.clear();
07406         res.content_length_ = 0;
07407         res.content_provider_ = nullptr;
07408         res.status = StatusCode::RangeNotSatisfiable_416;
07409         return write_response(strm, close_connection, req, res);
07410     }
07411
07412     return write_response_with_content(strm, close_connection, req, res);
07413 } else {
07414     if (res.status == -1) { res.status = StatusCode::NotFound_404; }
07415
07416     return write_response(strm, close_connection, req, res);
07417 }
07418 }
```

Граф вызовов:



Граф вызова функции:



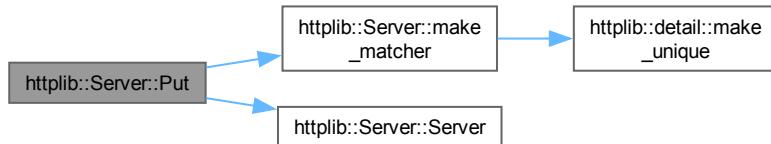
6.36.5.27 Put() [1/2]

```
Server & httpplib::Server::Put (
    const std::string & pattern,
    Handler handler) [inline]
```

См. определение в файле `httpplib.h` строка 6345

```
06345 {
06346     put_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06347     return *this;
06348 }
```

Граф вызовов:



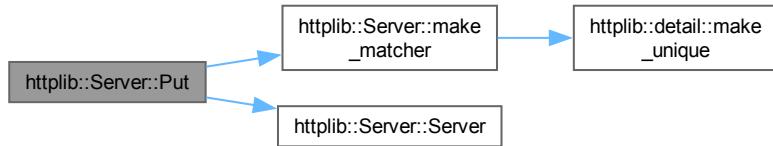
6.36.5.28 Put() [2/2]

```
Server & httpplib::Server::Put (
    const std::string & pattern,
    HandlerWithContentReader handler) [inline]
```

См. определение в файле `httpplib.h` строка 6350

```
06351 {
06352     put_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06353                                         std::move(handler));
06354     return *this;
06355 }
```

Граф вызовов:



6.36.5.29 read_content()

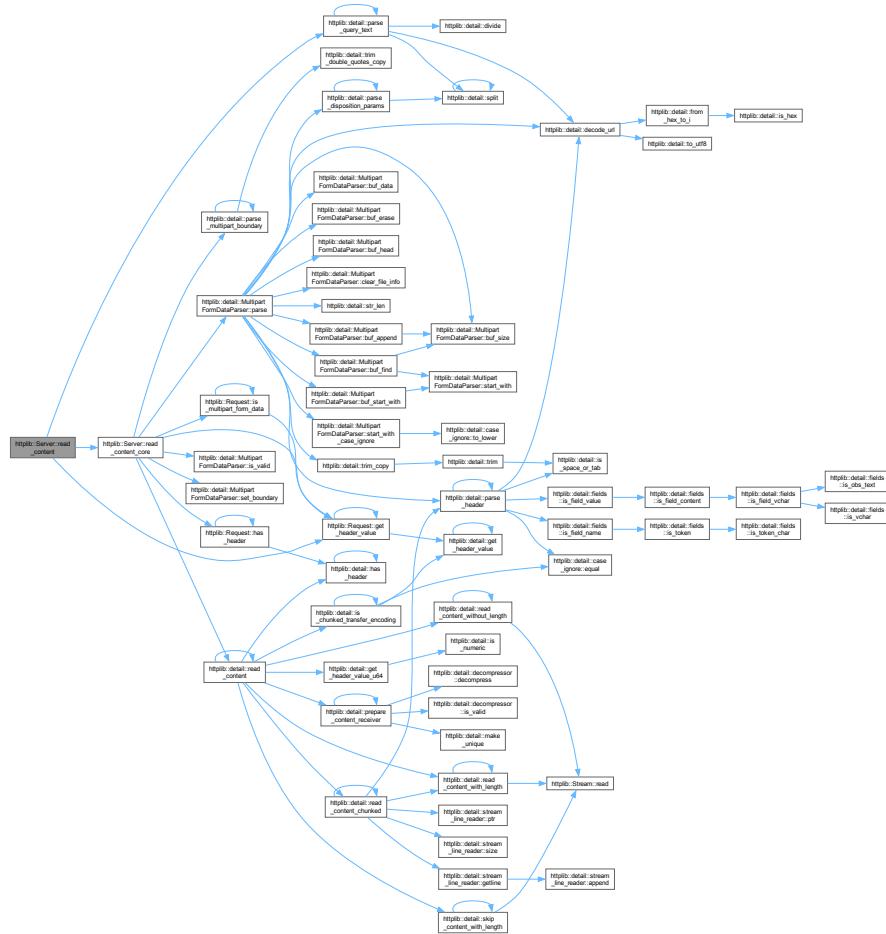
```
bool httplib::Server::read_content (
    Stream & strm,
    Request & req,
    Response & res) [inline], [private]
```

См. определение в файле [httplib.h](#) строка 6769

```

06769
06770     MultipartFormDataMap::iterator cur;
06771     auto file_count = 0;
06772     if (read_content_core(
06773         strm, req, res,
06774         // Regular
06775         [&](const char *buf, size_t n) {
06776             if (req.body.size() + n > req.body.max_size()) { return false; }
06777             req.body.append(buf, n);
06778             return true;
06779         },
06780         // Multipart
06781         [&](const MultipartFormData &file) {
06782             if (file_count++ == CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT) {
06783                 return false;
06784             }
06785             cur = req.files.emplace(file.name, file);
06786             return true;
06787         },
06788         [&](const char *buf, size_t n) {
06789             auto &content = cur->second.content;
06790             if (content.size() + n > content.max_size()) { return false; }
06791             content.append(buf, n);
06792             return true;
06793         }));
06794     const auto &content_type = req.get_header_value("Content-Type");
06795     if (!content_type.find("application/x-www-form-urlencoded")) {
06796         if (req.body.size() > CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH) {
06797             res.status = StatusCode::PayloadTooLarge_413; // NOTE: should be 414?
06798             return false;
06799         }
06800         detail::parse_query_text(req.body, req.params);
06801     }
06802     return true;
06803 }
06804 return false;
06805 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.30 `read_content_core()`

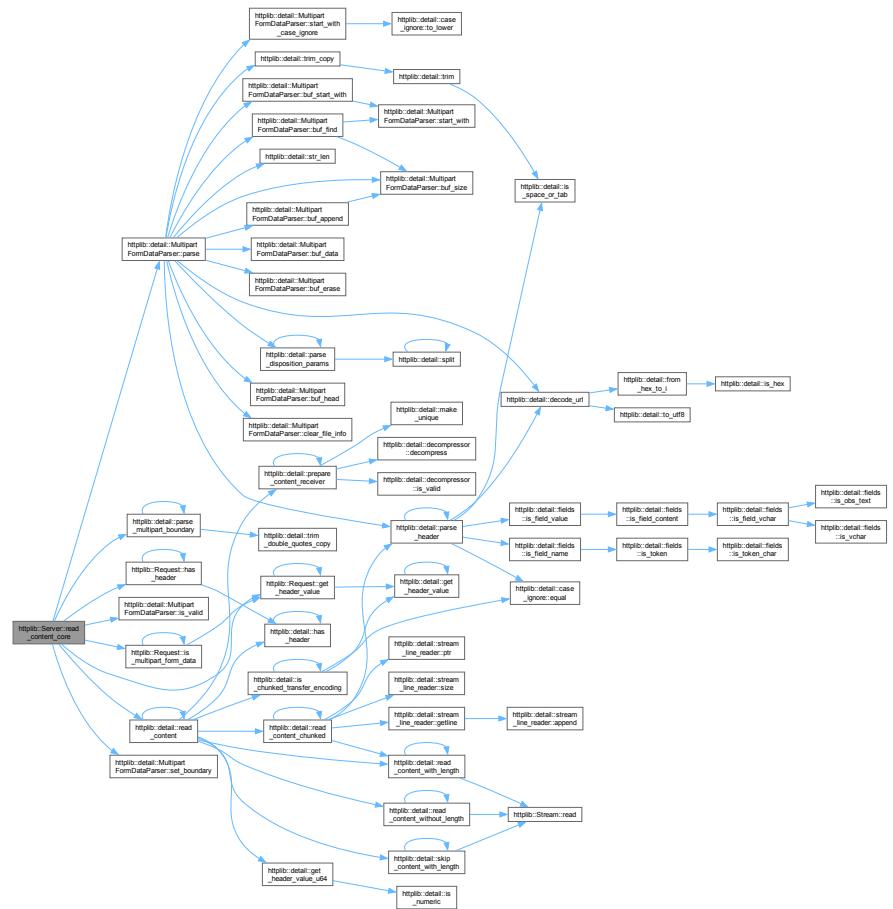
```

bool http://Server::read_content_core (
    Stream & strm,
    Request & req,
    Response & res,
    ContentReceiver receiver,
    MultipartContentHeader multipart_header,
    ContentReceiver multipart_receiver) const [inline], [private]
  
```

См. определение в файле [http://lib.h](#) строка 6817

```
06820                                     {
06821     detail::MultipartFormDataParser multipart_form_data_parser;
06822     ContentReceiverWithProgress out;
06823
06824     if (req.is_multipart_form_data()) {
06825         const auto &content_type = req.get_header_value("Content-Type");
06826         std::string boundary;
06827         if (!detail::parse_multipart_boundary(content_type, boundary)) {
06828             res.status = StatusCode::BadRequest_400;
06829             return false;
06830         }
06831
06832         multipart_form_data_parser.set_boundary(std::move(boundary));
06833         out = [&](const char *buf, size_t n, uint64_t /*off*/, uint64_t /*len*/) {
06834             /* For debug
06835             size_t pos = 0;
06836             while (pos < n) {
06837                 auto read_size = (std::min)(size_t)(1, n - pos);
06838                 auto ret = multipart_form_data_parser.parse(
06839                     buf + pos, read_size, multipart_receiver, multipart_header);
06840                 if (!ret) { return false; }
06841                 pos += read_size;
06842             }
06843             return true;
06844         */
06845         return multipart_form_data_parser.parse(buf, n, multipart_receiver,
06846                                         multipart_header);
06847     };
06848 } else {
06849     out = [receiver](const char *buf, size_t n, uint64_t /*off*/,
06850                      uint64_t /*len*/) { return receiver(buf, n); };
06851 }
06852
06853 if (req.method == "DELETE" && !req.has_header("Content-Length")) {
06854     return true;
06855 }
06856
06857 if (!detail::read_content(strm, req, payload_max_length_, res.status, nullptr,
06858                             out, true)) {
06859     return false;
06860 }
06861
06862 if (req.is_multipart_form_data()) {
06863     if (!multipart_form_data_parser.is_valid()) {
06864         res.status = StatusCode::BadRequest_400;
06865         return false;
06866     }
06867 }
06868
06869 return true;
06870 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.31 read_content_with_content_receiver()

```
bool httplib::Server::read_content_with_content_receiver (
```

`Stream & strm,`

`Request & req,`

`Response & res,`

`ContentReceiver receiver,`

`MultipartContentHeader multipart_header,`

`ContentReceiver multipart_receiver)` [inline], [private]

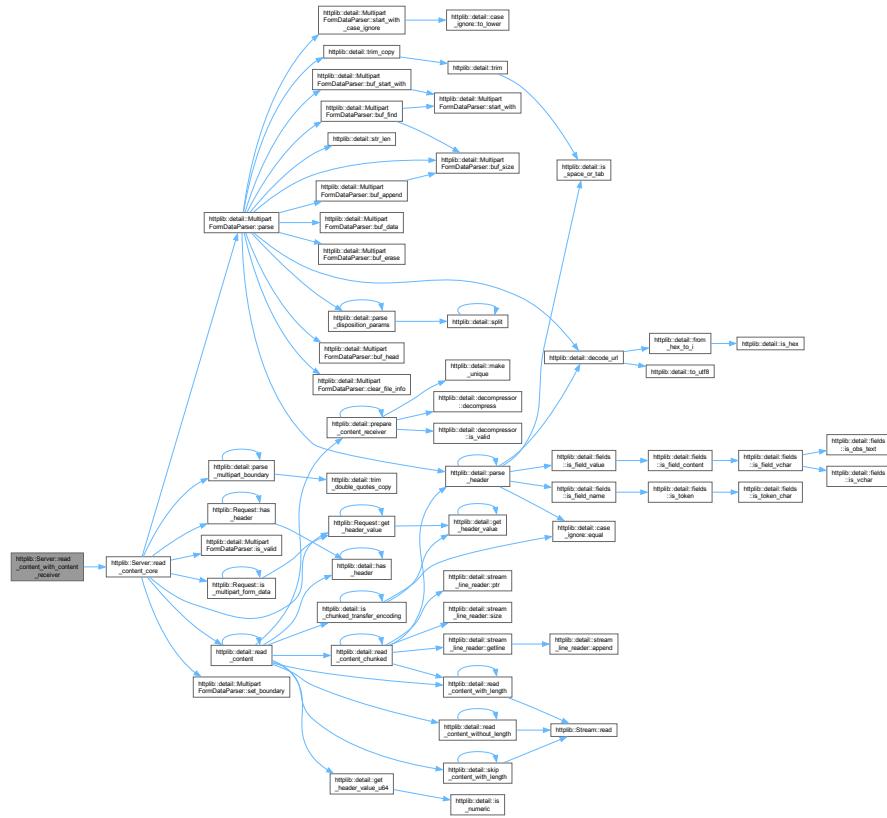
См. определение в файле `httpplib.h` строка 6807

```

06810
06811  return read_content_core(strm, req, res, std::move(receiver),
06812          std::move(multipart_header),
06813          std::move(multipart_receiver));
06814 }

```

Граф вызовов:



Граф вызова функции:



6.36.5.32 `remove_mount_point()`

```

bool httpplib::Server::remove_mount_point (
    const std::string & mount_point) [inline]

```

См. определение в файле `httpplib.h` строка 6404

```

06404
06405  for (auto it = base_dirs_.begin(); it != base_dirs_.end(); ++it) {
06406      if (it->mount_point == mount_point) {
06407          base_dirs_.erase(it);
06408          return true;
06409      }
06410  }
06411  return false;
06412 }

```

6.36.5.33 `routing()`

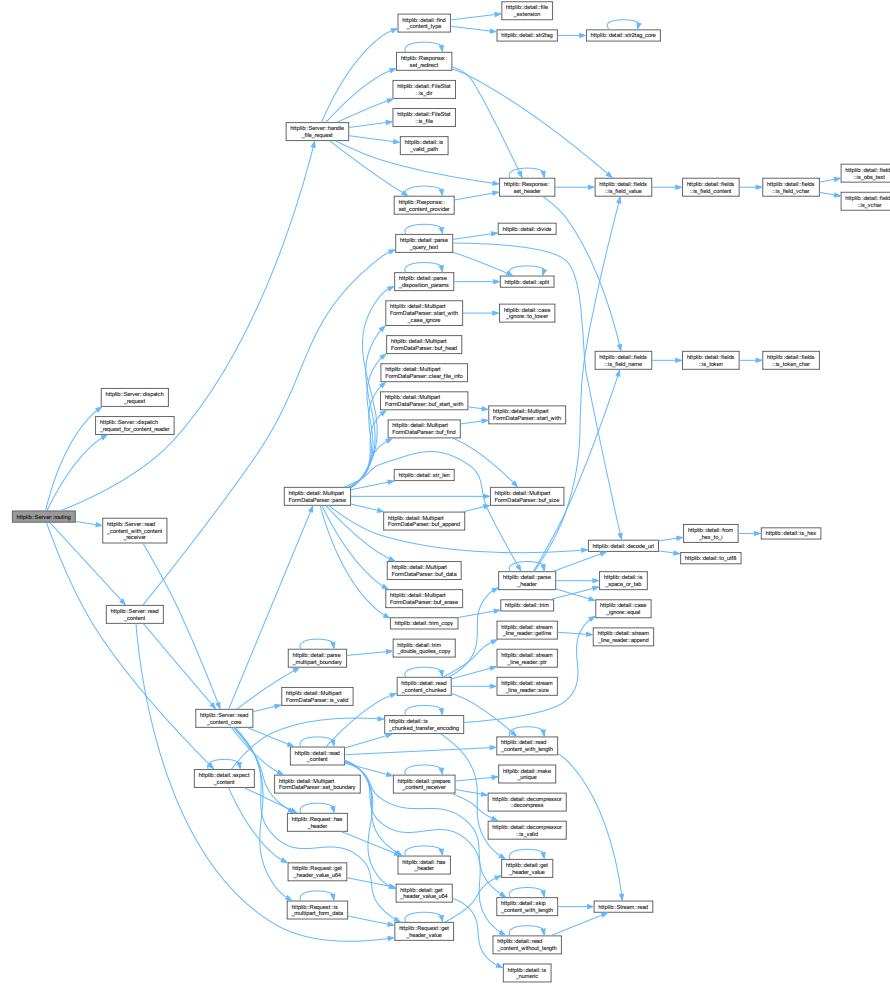
```
bool httpplib::Server::routing (
    Request & req,
    Response & res,
    Stream & strm) [inline], [private]
```

См. определение в файле `httpplib.h` строка 7033

```
07033     if (pre_routing_handler_ &&
07034         pre_routing_handler_(req, res) == HandlerResponse::Handled) {
07035             return true;
07036         }
07037     }
07038
07039     // File handler
07040     auto is_head_request = req.method == "HEAD";
07041     if ((req.method == "GET" || is_head_request) &&
07042         handle_file_request(req, res, is_head_request)) {
07043         return true;
07044     }
07045
07046     if (detail::expect_content(req)) {
07047         // Content reader handler
07048     {
07049         ContentReader reader(
07050             [&](ContentReceiver receiver) {
07051                 return read_content_with_content_receiver(
07052                     strm, req, res, std::move(receiver), nullptr, nullptr);
07053             },
07054             [&](MultipartContentHeader header, ContentReceiver receiver) {
07055                 return read_content_with_content_receiver(strm, req, res, nullptr,
07056                     std::move(header),
07057                     std::move(receiver));
07058         );
07059
07060         if (req.method == "POST") {
07061             if (dispatch_request_for_content_reader(
07062                 req, res, std::move(reader),
07063                 post_handlers_for_content_reader_)) {
07064                 return true;
07065             }
07066         } else if (req.method == "PUT") {
07067             if (dispatch_request_for_content_reader(
07068                 req, res, std::move(reader),
07069                 put_handlers_for_content_reader_)) {
07070                 return true;
07071             }
07072         } else if (req.method == "PATCH") {
07073             if (dispatch_request_for_content_reader(
07074                 req, res, std::move(reader),
07075                 patch_handlers_for_content_reader_)) {
07076                 return true;
07077             }
07078         } else if (req.method == "DELETE") {
07079             if (dispatch_request_for_content_reader(
07080                 req, res, std::move(reader),
07081                 delete_handlers_for_content_reader_)) {
07082                 return true;
07083             }
07084         }
07085     }
07086
07087     // Read content into `req.body`
07088     if (!read_content(strm, req, res)) { return false; }
07089 }
07090
07091     // Regular handler
07092     if (req.method == "GET" || req.method == "HEAD") {
07093         return dispatch_request(req, res, get_handlers_);
07094     } else if (req.method == "POST") {
07095         return dispatch_request(req, res, post_handlers_);
07096     } else if (req.method == "PUT") {
07097         return dispatch_request(req, res, put_handlers_);
07098     } else if (req.method == "DELETE") {
07099         return dispatch_request(req, res, delete_handlers_);
07100     } else if (req.method == "OPTIONS") {
07101         return dispatch_request(req, res, options_handlers_);
07102     } else if (req.method == "PATCH") {
07103         return dispatch_request(req, res, patch_handlers_);
07104     }
07105
07106     res.status = StatusCode::BadRequest_400;
07107     return false;
```

07108 }

Граф вызовов:



Граф вызова функции:



6.36.5.34 set_address_family()

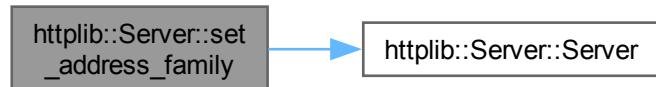
```
Server & httplib::Server::set_address_family (
    int family) [inline]
```

См. определение в файле [httplib.h](#) строка 6472

```

06472 {
06473   address_family_ = family;
06474   return *this;
06475 }
```

Граф вызовов:



6.36.5.35 `set_base_dir()`

```

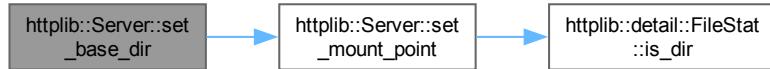
bool httpplib::Server::set_base_dir (
    const std::string & dir,
    const std::string & mount_point = std::string() [inline]
```

См. определение в файле [httpplib.h](#) строка 6386

```

06387 {
06388   return set_mount_point(mount_point, dir);
06389 }
```

Граф вызовов:



6.36.5.36 `set_default_file_mimetype()`

```

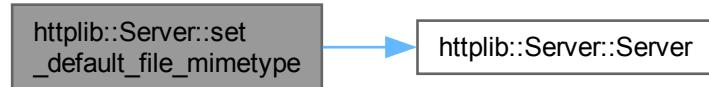
Server & httpplib::Server::set_default_file_mimetype (
    const std::string & mime) [inline]
```

См. определение в файле [httpplib.h](#) строка 6421

```

06421 {
06422   default_file_mimetype_ = mime;
06423   return *this;
06424 }
```

Граф вызовов:



6.36.5.37 set_default_headers()

```
Server & httplib::Server::set_default_headers (\n    Headers headers) [inline]
```

См. определение в файле [httplib.h](#) строка 6492

```
06492 {\n06493     default_headers_ = std::move(headers);\n06494     return *this;\n06495 }
```

Граф вызовов:



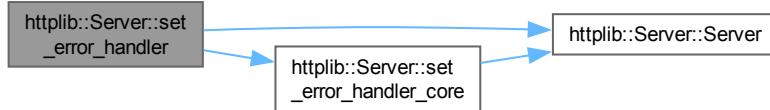
6.36.5.38 set_error_handler()

```
template<class ErrorHandlerFunc>\nServer & httplib::Server::set_error_handler (\n    ErrorHandlerFunc && handler) [inline]
```

См. определение в файле [httplib.h](#) строка 996

```
00996 {\n00997     return set_error_handler_core(\n00998         std::forward<ErrorHandlerFunc>(handler),\n00999         std::is_convertible<ErrorHandlerFunc, HandlerWithResponse>{});\n01000 }
```

Граф вызовов:



6.36.5.39 set_error_handler_core() [1/2]

```
Server & httplib::Server::set_error_handler_core (
    Handler handler,
    std::false_type ) [inline], [private]
```

См. определение в файле [httplib.h](#) строка 6437

```
06438     {
06439     error_handler_ = [handler](const Request &req, Response &res) {
06440         handler(req, res);
06441         return HandlerResponse::Handled;
06442     };
06443     return *this;
06444 }
```

Граф вызовов:



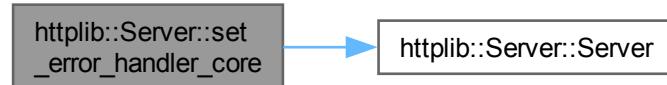
6.36.5.40 set_error_handler_core() [2/2]

```
Server & httplib::Server::set_error_handler_core (
    HandlerWithResponse handler,
    std::true_type ) [inline], [private]
```

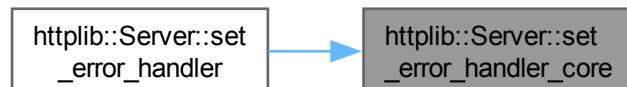
См. определение в файле [httplib.h](#) строка 6431

```
06432     {
06433     error_handler_ = std::move(handler);
06434     return *this;
06435 }
```

Граф вызовов:



Граф вызова функции:



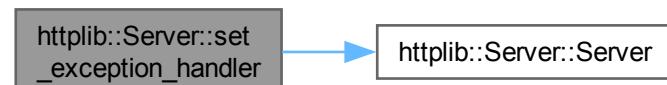
6.36.5.41 set_exception_handler()

```
Server & httplib::Server::set_exception_handler (  
    ExceptionHandler handler) [inline]
```

См. определение в файле [httplib.h](#) строка 6446

```
06446 {  
06447     exception_handler_ = std::move(handler);  
06448     return *this;  
06449 }
```

Граф вызовов:



6.36.5.42 `set_expect_100_continue_handler()`

```
Server & httpplib::Server::set_expect_100_continue_handler (
    Expect100ContinueHandler handler) [inline]
```

См. определение в файле `httpplib.h` строка 6467

```
06467 {
06468     expect_100_continue_handler_ = std::move(handler);
06469     return *this;
06470 }
```

6.36.5.43 `set_file_extension_and_mimetype_mapping()`

```
Server & httpplib::Server::set_file_extension_and_mimetype_mapping (
    const std::string & ext,
    const std::string & mime) [inline]
```

См. определение в файле `httpplib.h` строка 6415

```
06416 {
06417     file_extension_and_mimetype_map_[ext] = mime;
06418     return *this;
06419 }
```

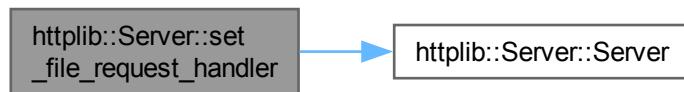
6.36.5.44 `set_file_request_handler()`

```
Server & httpplib::Server::set_file_request_handler (
    Handler handler) [inline]
```

См. определение в файле `httpplib.h` строка 6426

```
06426 {
06427     file_request_handler_ = std::move(handler);
06428     return *this;
06429 }
```

Граф вызовов:



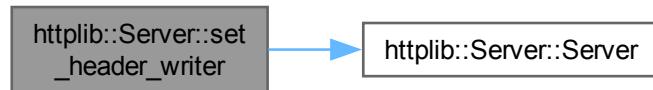
6.36.5.45 `set_header_writer()`

```
Server & httpplib::Server::set_header_writer (
    std::function< ssize_t(Stream &, Headers &) > const & writer) [inline]
```

См. определение в файле [httpplib.h](#) строка [6497](#)

```
06498
06499     header_writer_ = writer;
06500     return *this;
06501 }
```

Граф вызовов:

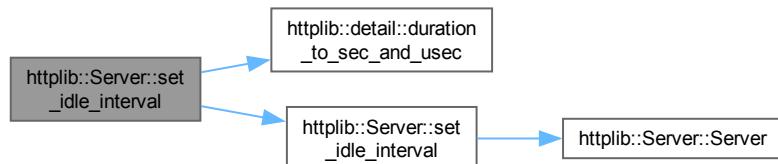
6.36.5.46 `set_idle_interval()` [1/2]

```
template<class Rep, class Period>
Server & httpplib::Server::set_idle_interval (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка [2248](#)

```
02248
02249     detail::duration_to_sec_and_usec(
02250         duration, [&](time_t sec, time_t usec) { set_idle_interval(sec, usec); });
02251     return *this;
02252 }
```

Граф вызовов:



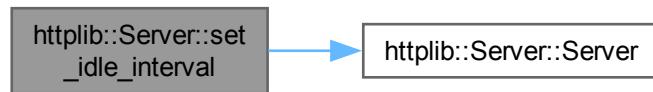
6.36.5.47 `set_idle_interval()` [2/2]

```
Server & httpplib::Server::set_idle_interval (
    time_t sec,
    time_t usec = 0) [inline]
```

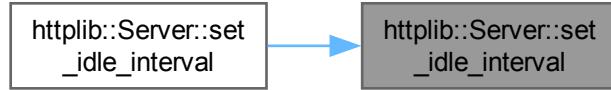
См. определение в файле `httpplib.h` строка 6525

```
06525 {
06526     idle_interval_sec_ = sec;
06527     idle_interval_usec_ = usec;
06528     return *this;
06529 }
```

Граф вызовов:



Граф вызова функции:

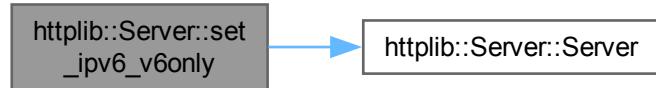
6.36.5.48 `set_ipv6_v6only()`

```
Server & httpplib::Server::set_ipv6_v6only (
    bool on) [inline]
```

См. определение в файле `httpplib.h` строка 6482

```
06482 {
06483     ipv6_v6only_ = on;
06484     return *this;
06485 }
```

Граф вызовов:



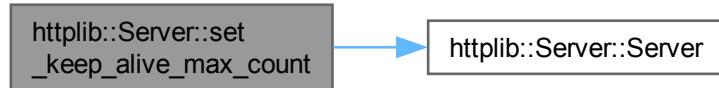
6.36.5.49 `set_keep_alive_max_count()`

`Server & httpplib::Server::set_keep_alive_max_count (\n size_t count) [inline]`

См. определение в файле [httpplib.h](#) строка 6503

```
06503 {\n06504     keep_alive_max_count_ = count;\n06505     return *this;\n06506 }
```

Граф вызовов:



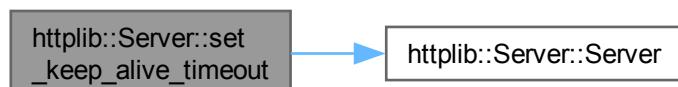
6.36.5.50 `set_keep_alive_timeout()`

`Server & httpplib::Server::set_keep_alive_timeout (\n time_t sec) [inline]`

См. определение в файле [httpplib.h](#) строка 6508

```
06508 {\n06509     keep_alive_timeout_sec_ = sec;\n06510     return *this;\n06511 }
```

Граф вызовов:



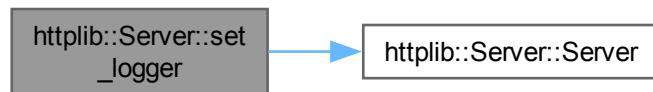
6.36.5.51 `set_logger()`

```
Server & httpplib::Server::set_logger (
    Logger logger) [inline]
```

См. определение в файле `httpplib.h` строка 6461

```
06461
06462     logger_ = std::move(logger);
06463     return *this;
06464 }
```

Граф вызовов:



Граф вызова функции:

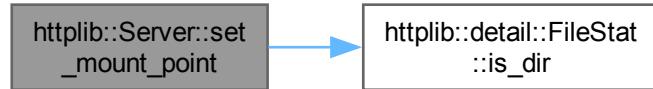
6.36.5.52 `set_mount_point()`

```
bool httpplib::Server::set_mount_point (
    const std::string & mount_point,
    const std::string & dir,
    Headers headers = Headers()) [inline]
```

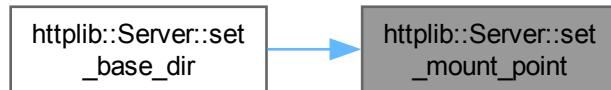
См. определение в файле `httpplib.h` строка 6391

```
06392
06393     detail::FileStat stat(dir);
06394     if (stat.is_dir()) {
06395         std::string mnt = !mount_point.empty() ? mount_point : "/";
06396         if (!mnt.empty() && mnt[0] == '/') {
06397             base_dirs_.push_back({mnt, dir, std::move(headers)});
06398         }
06399     }
06400 }
06401 return false;
06402 }
```

Граф вызовов:



Граф вызова функции:



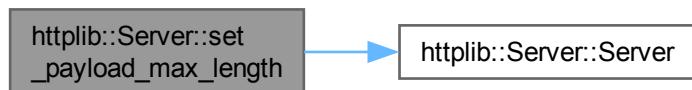
6.36.5.53 set_payload_max_length()

`Server & http://Server::set_payload_max_length (size_t length) [inline]`

См. определение в файле [http://Server.h](#) строка [6531](#)

```
06531 {  
06532     payload_max_length_ = length;  
06533     return *this;  
06534 }
```

Граф вызовов:



6.36.5.54 `set_post_routing_handler()`

```
Server & httpplib::Server::set_post_routing_handler (
    Handler handler) [inline]
```

См. определение в файле `httpplib.h` строка 6456

```
06456 {
06457     post_routing_handler_ = std::move(handler);
06458     return *this;
06459 }
```

Граф вызовов:

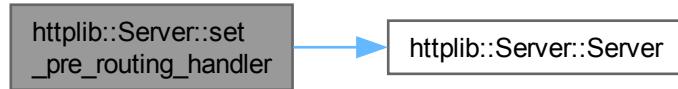
6.36.5.55 `set_pre_routing_handler()`

```
Server & httpplib::Server::set_pre_routing_handler (
    HandlerWithResponse handler) [inline]
```

См. определение в файле `httpplib.h` строка 6451

```
06451 {
06452     pre_routing_handler_ = std::move(handler);
06453     return *this;
06454 }
```

Граф вызовов:



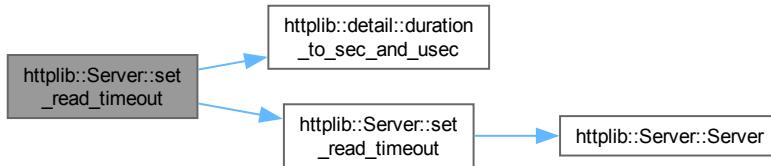
6.36.5.56 `set_read_timeout()` [1/2]

```
template<class Rep, class Period>
Server & httpplib::Server::set_read_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2232

```
02232 {
02233     detail::duration_to_sec_and_usec(
02234         duration, [&](time_t sec, time_t usec) { set_read_timeout(sec, usec); });
02235     return *this;
02236 }
```

Граф вызовов:

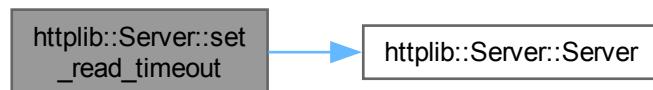
6.36.5.57 `set_read_timeout()` [2/2]

```
Server & httpplib::Server::set_read_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

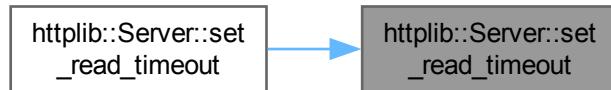
См. определение в файле [httpplib.h](#) строка 6513

```
06513 {
06514     read_timeout_sec_ = sec;
06515     read_timeout_usec_ = usec;
06516     return *this;
06517 }
```

Граф вызовов:



Граф вызова функции:



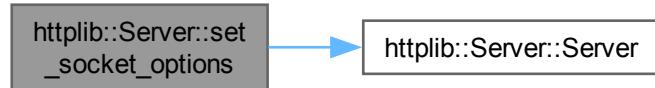
6.36.5.58 `set_socket_options()`

```
Server & httpplib::Server::set_socket_options (\n    SocketOptions socket_options) [inline]
```

См. определение в файле [httpplib.h](#) строка 6487

```
06487\n06488     socket_options_ = std::move(socket_options);\n06489     return *this;\n06490 }
```

Граф вызовов:



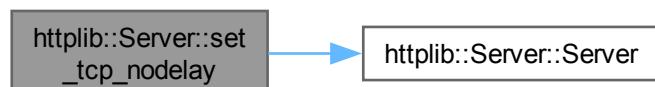
6.36.5.59 `set_tcp_nodelay()`

```
Server & httpplib::Server::set_tcp_nodelay (\n    bool on) [inline]
```

См. определение в файле [httpplib.h](#) строка 6477

```
06477\n06478     tcp_nodelay_ = on;\n06479     return *this;\n06480 }
```

Граф вызовов:



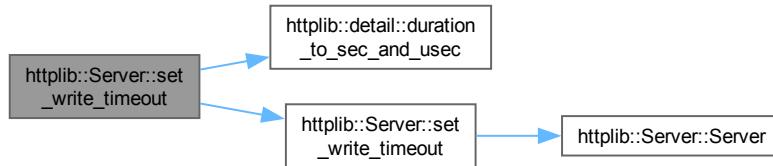
6.36.5.60 `set_write_timeout()` [1/2]

```
template<class Rep, class Period>
Server & httpplib::Server::set_write_timeout (
    const std::chrono::duration< Rep, Period > & duration) [inline]
```

См. определение в файле [httpplib.h](#) строка 2240

```
02240 {
02241     detail::duration_to_sec_and_usec(
02242         duration, [&](time_t sec, time_t usec) { set_write_timeout(sec, usec); });
02243     return *this;
02244 }
```

Граф вызовов:

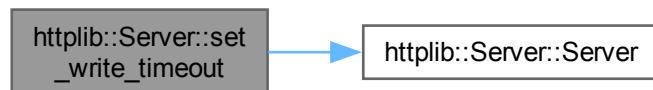
6.36.5.61 `set_write_timeout()` [2/2]

```
Server & httpplib::Server::set_write_timeout (
    time_t sec,
    time_t usec = 0) [inline]
```

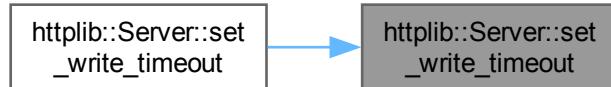
См. определение в файле [httpplib.h](#) строка 6519

```
06519 {
06520     write_timeout_sec_ = sec;
06521     write_timeout_usec_ = usec;
06522     return *this;
06523 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.62 `stop()`

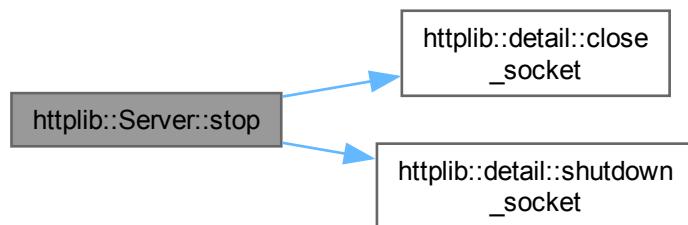
`void httpplib::Server::stop () [inline]`

См. определение в файле [httpplib.h](#) строка 6563

```

06563     if (is_running_) {
06564         if (svr_sock_ != INVALID_SOCKET) {
06565             assert(svr_sock_ != INVALID_SOCKET);
06566             std::atomic<socket_t> sock(svr_sock_).exchange(INVALID_SOCKET));
06567             detail::shutdown_socket(sock);
06568             detail::close_socket(sock);
06569         }
06570         is_decommissioned = false;
06571     }
  
```

Граф вызовов:



6.36.5.63 `wait_until_ready()`

`void httpplib::Server::wait_until_ready () const [inline]`

См. определение в файле [httpplib.h](#) строка 6557

```

06557     {
06558         while (!is_running_ && !is_decommissioned) {
06559             std::this_thread::sleep_for(std::chrono::milliseconds{1});
06560         }
06561     }
  
```

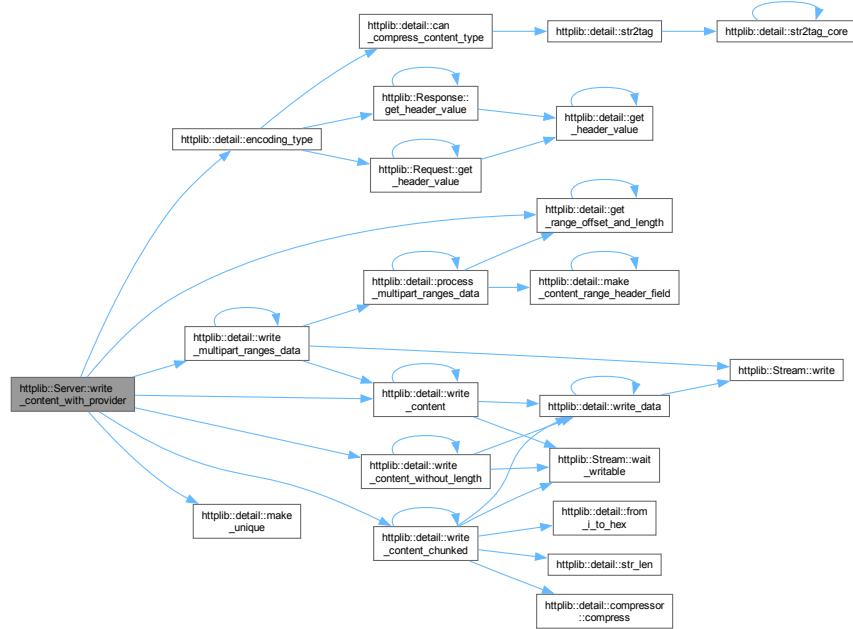
6.36.5.64 `write_content_with_provider()`

```
bool httpplib::Server::write_content_with_provider (
    Stream & strm,
    const Request & req,
    Response & res,
    const std::string & boundary,
    const std::string & content_type) [inline], [private]
```

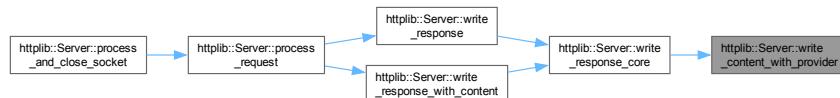
См. определение в файле `httpplib.h` строка 6715

```
06717     auto is_shutting_down = [this]() {
06718         return this->svr_sock_ == INVALID_SOCKET;
06719     };
06720 
06721 
06722     if (res.content_length_ > 0) {
06723         if (req.ranges.empty()) {
06724             return detail::write_content(strm, res.content_provider_, 0,
06725                                         res.content_length_, is_shutting_down);
06726         } else if (req.ranges.size() == 1) {
06727             auto offset_and_length = detail::get_range_offset_and_length(
06728                 req.ranges[0], res.content_length_);
06729 
06730             return detail::write_content(strm, res.content_provider_,
06731                                         offset_and_length.first,
06732                                         offset_and_length.second, is_shutting_down);
06733         } else {
06734             return detail::write_multipart_ranges_data(
06735                 strm, req, res, boundary, content_type, res.content_length_,
06736                 is_shutting_down);
06737         }
06738     } else {
06739         if (res.is_chunked_content_provider_) {
06740             auto type = detail::encoding_type(req, res);
06741 
06742             std::unique_ptr<detail::compressor> compressor;
06743             if (type == detail::EncodingType::Gzip) {
06744 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
06745                 compressor = detail::make_unique<detail::gzip_compressor>();
06746 #endif
06747             } else if (type == detail::EncodingType::Brotli) {
06748 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
06749                 compressor = detail::make_unique<detail::brotli_compressor>();
06750 #endif
06751             } else if (type == detail::EncodingType::Zstd) {
06752 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
06753                 compressor = detail::make_unique<detail::zstd_compressor>();
06754 #endif
06755             } else {
06756                 compressor = detail::make_unique<detail::nocompressor>();
06757             }
06758             assert(compressor != nullptr);
06759 
06760             return detail::write_content_chunked(strm, res.content_provider_,
06761                                                 is_shutting_down, *compressor);
06762         } else {
06763             return detail::write_content_without_length(strm, res.content_provider_,
06764                                                 is_shutting_down);
06765         }
06766     }
06767 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.65 `write_response()`

```

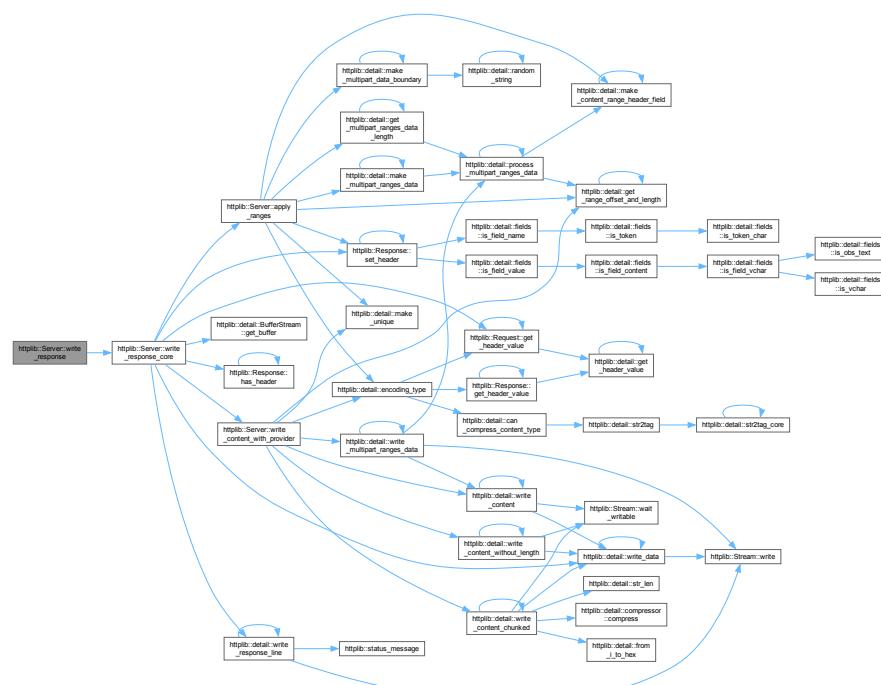
bool httpplib::Server::write_response (
    Stream & strm,
    bool close_connection,
    Request & req,
    Response & res) [inline], [private]
    
```

См. определение в файле `httpplib.h` строка 6625

```

06626 {
06627 // NOTE: `req.ranges` should be empty, otherwise it will be applied
06628 // incorrectly to the error content.
06629 req.ranges.clear();
06630 return write_response_core(strm, close_connection, req, res, false);
06631 }
    
```

Граф вызовов:



Граф вызова функции:



6.36.5.66 `write_response_core()`

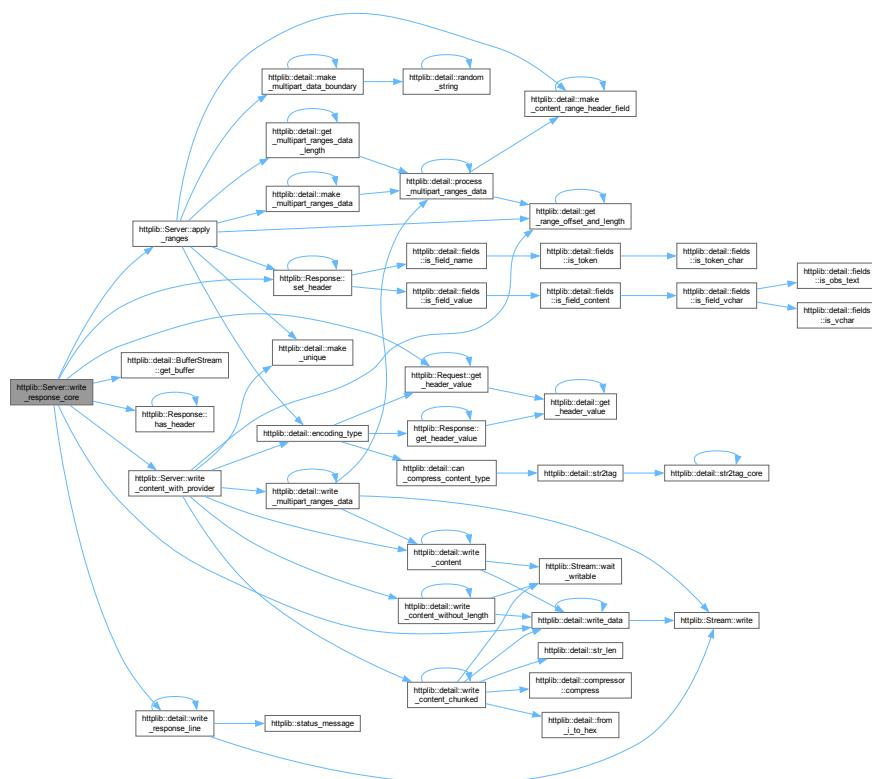
```
bool http://Server::write_response_core (
    Stream & strm,
    bool close_connection,
    const Request & req,
    Response & res,
    bool need_apply_ranges) [inline], [private]
```

См. определение в файле `http://Server.h` строка 6640

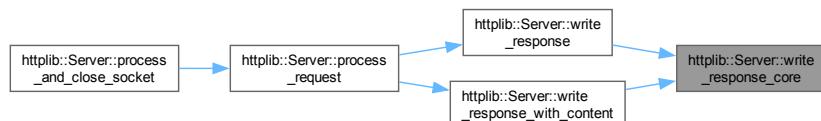
```
06642     assert(res.status != -1);
06643
06644
06645     if (400 <= res.status && error_handler_ &&
06646         error_handler_(req, res) == HandlerResponse::Handled) {
06647         need_apply_ranges = true;
06648     }
06649
06650     std::string content_type;
06651     std::string boundary;
```

```
06652 if (need_apply_ranges) { apply_ranges(req, res, content_type, boundary); }
06653
06654 // Prepare additional headers
06655 if (close_connection || req.get_header_value("Connection") == "close") {
06656     res.set_header("Connection", "close");
06657 } else {
06658     std::string s = "timeout=";
06659     s += std::to_string(keep_alive_timeout_sec_);
06660     s += ", max=";
06661     s += std::to_string(keep_alive_max_count_);
06662     res.set_header("Keep-Alive", s);
06663 }
06664
06665 if ((!res.body.empty() || res.content_length_ > 0 || res.content_provider_) &&
06666     !res.has_header("Content-Type")) {
06667     res.set_header("Content-Type", "text/plain");
06668 }
06669
06670 if (res.body.empty() && !res.content_length_ && !res.content_provider_ &&
06671     !res.has_header("Content-Length")) {
06672     res.set_header("Content-Length", "0");
06673 }
06674
06675 if (req.method == "HEAD" && !res.has_header("Accept-Ranges")) {
06676     res.set_header("Accept-Ranges", "bytes");
06677 }
06678
06679 if (post_routing_handler_) { post_routing_handler_(req, res); }
06680
06681 // Response line and headers
06682 {
06683     detail::BufferStream bstrm;
06684     if (!detail::write_response_line(bstrm, res.status)) { return false; }
06685     if (!header_writer_(bstrm, res.headers)) { return false; }
06686
06687 // Flush buffer
06688 auto &data = bstrm.get_buffer();
06689 detail::write_data(strm, data.data(), data.size());
06690 }
06691
06692 // Body
06693 auto ret = true;
06694 if (req.method != "HEAD") {
06695     if (!res.body.empty()) {
06696         if (!detail::write_data(strm, res.body.data(), res.body.size())) {
06697             ret = false;
06698         }
06699     } else if (res.content_provider_) {
06700         if (write_content_with_provider(strm, req, res, boundary, content_type)) {
06701             res.content_provider_success_ = true;
06702         } else {
06703             ret = false;
06704         }
06705     }
06706 }
06707
06708 // Log
06709 if (logger_) { logger_(req, res); }
06710
06711 return ret;
06712 }
```

Граф вызовов:



Граф вызова функции:



6.36.5.67 write_response_with_content()

```

bool http://Server::write_response_with_content (
    Stream & strm,
    bool close_connection,
    const Request & req,
    Response & res) [inline], [private]
  
```

См. определение в файле [http://lib.h](#) строка 6633

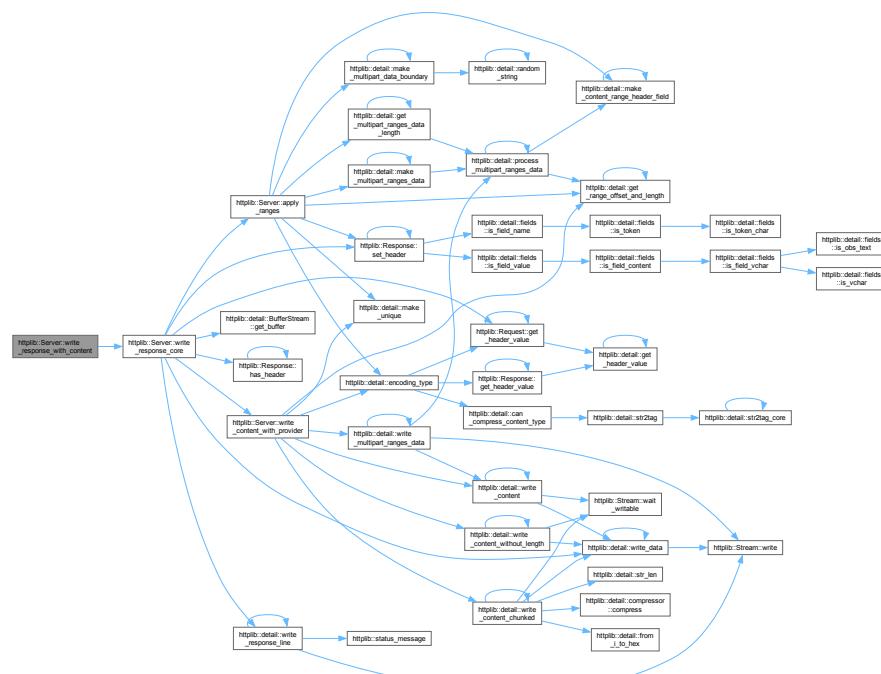
```

06636 {  

06637     return write_response_core(strm, close_connection, req, res, true);  

06638 }
  
```

Граф вызовов:



Граф вызова функции:



6.36.6 Данные класса

6.36.6.1 address_family_

```
int httpplib::Server::address_family_ = AF_UNSPEC [private]
```

См. определение в файле [httpplib.h](#) строка 1152

6.36.6.2 base_dirs_

```
std::vector<MountPointEntry> httpplib::Server::base_dirs_ [private]
```

См. определение в файле [httpplib.h](#) строка 1128

6.36.6.3 `default_file_mimetype_`

`std::string httpplib::Server::default_file_mimetype_ = "application/octet-stream" [private]`

См. определение в файле [httpplib.h](#) строка [1130](#)

6.36.6.4 `default_headers_`

`Headers httpplib::Server::default_headers_ [private]`

См. определение в файле [httpplib.h](#) строка [1157](#)

6.36.6.5 `delete_handlers_`

`Handlers httpplib::Server::delete_handlers_ [private]`

См. определение в файле [httpplib.h](#) строка [1140](#)

6.36.6.6 `delete_handlers_for_content_reader_`

`HandlersForContentReader httpplib::Server::delete_handlers_for_content_reader_ [private]`

См. определение в файле [httpplib.h](#) строка [1141](#)

6.36.6.7 `error_handler_`

`HandlerWithResponse httpplib::Server::error_handler_ [private]`

См. определение в файле [httpplib.h](#) строка [1144](#)

6.36.6.8 `exception_handler_`

`ExceptionHandler httpplib::Server::exception_handler_ [private]`

См. определение в файле [httpplib.h](#) строка [1145](#)

6.36.6.9 `expect_100_continue_handler_`

`Expect100ContinueHandler httpplib::Server::expect_100_continue_handler_ [private]`

См. определение в файле [httpplib.h](#) строка [1148](#)

6.36.6.10 `file_extension_and_mimetype_map_`

`std::map<std::string, std::string> httpplib::Server::file_extension_and_mimetype_map_ [private]`

См. определение в файле [httpplib.h](#) строка [1129](#)

6.36.6.11 `file_request_handler_`

`Handler httpplib::Server::file_request_handler_ [private]`

См. определение в файле [httpplib.h](#) строка [1131](#)

6.36.6.12 `get_handlers_`

`Handlers httpplib::Server::get_handlers_ [private]`

См. определение в файле [httpplib.h](#) строка [1133](#)

6.36.6.13 `header_writer_`

`std::function<ssize_t(Stream &, Headers &)> httpplib::Server::header_writer_ [private]`

Инициализатор

=
 `detail::write_headers`

См. определение в файле [httpplib.h](#) строка [1158](#)

6.36.6.14 `idle_interval_sec_`

`time_t httpplib::Server::idle_interval_sec_ = 0 [protected]`

См. определение в файле [httpplib.h](#) строка [1062](#)

6.36.6.15 `idle_interval_usec_`

`time_t httpplib::Server::idle_interval_usec_ = 0 [protected]`

См. определение в файле [httpplib.h](#) строка [1063](#)

6.36.6.16 `ipv6_v6only_`

`bool httpplib::Server::ipv6_v6only_ = false [private]`

См. определение в файле [httpplib.h](#) строка [1154](#)

6.36.6.17 `is_decommissioned`

`std::atomic<bool> httpplib::Server::is_decommissioned {false} [private]`

См. определение в файле [httpplib.h](#) строка [1121](#)
`01121 {false};`

6.36.6.18 `is_running_`

`std::atomic<bool> httpplib::Server::is_running_ {false} [private]`

См. определение в файле [httpplib.h](#) строка [1120](#)
01120 {`false`};

6.36.6.19 `keep_alive_max_count_`

`size_t httpplib::Server::keep_alive_max_count_ = 100 [protected]`

См. определение в файле [httpplib.h](#) строка [1056](#)

6.36.6.20 `keep_alive_timeout_sec_`

`time_t httpplib::Server::keep_alive_timeout_sec_ = 5 [protected]`

См. определение в файле [httpplib.h](#) строка [1057](#)

6.36.6.21 `logger_`

`Logger httpplib::Server::logger_ [private]`

См. определение в файле [httpplib.h](#) строка [1150](#)

6.36.6.22 `new_task_queue`

`std::function<TaskQueue *(void)> httpplib::Server::new_task_queue`

См. определение в файле [httpplib.h](#) строка [1046](#)

6.36.6.23 `options_handlers_`

`Handlers httpplib::Server::options_handlers_ [private]`

См. определение в файле [httpplib.h](#) строка [1142](#)

6.36.6.24 `patch_handlers_`

`Handlers httpplib::Server::patch_handlers_ [private]`

См. определение в файле [httpplib.h](#) строка [1138](#)

6.36.6.25 `patch_handlers_for_content_reader_`

`HandlersForContentReader httpplib::Server::patch_handlers_for_content_reader_ [private]`

См. определение в файле [httpplib.h](#) строка [1139](#)

6.36.6.26 `payload_max_length_`

`size_t httpplib::Server::payload_max_length_ = ((std::numeric_limits<size_t>::max)())` [protected]

См. определение в файле [httpplib.h](#) строка [1064](#)

6.36.6.27 `post_handlers_`

`Handlers httpplib::Server::post_handlers_` [private]

См. определение в файле [httpplib.h](#) строка [1134](#)

6.36.6.28 `post_handlers_for_content_reader_`

`HandlersForContentReader httpplib::Server::post_handlers_for_content_reader_` [private]

См. определение в файле [httpplib.h](#) строка [1135](#)

6.36.6.29 `post_routing_handler_`

`Handler httpplib::Server::post_routing_handler_` [private]

См. определение в файле [httpplib.h](#) строка [1147](#)

6.36.6.30 `pre_routing_handler_`

`HandlerWithResponse httpplib::Server::pre_routing_handler_` [private]

См. определение в файле [httpplib.h](#) строка [1146](#)

6.36.6.31 `put_handlers_`

`Handlers httpplib::Server::put_handlers_` [private]

См. определение в файле [httpplib.h](#) строка [1136](#)

6.36.6.32 `put_handlers_for_content_reader_`

`HandlersForContentReader httpplib::Server::put_handlers_for_content_reader_` [private]

См. определение в файле [httpplib.h](#) строка [1137](#)

6.36.6.33 `read_timeout_sec_`

`time_t httpplib::Server::read_timeout_sec_ = 5` [protected]

См. определение в файле [httpplib.h](#) строка [1058](#)

6.36.6.34 `read_timeout_usec_`

`time_t httpplib::Server::read_timeout_usec_ = 0` [protected]

См. определение в файле [httpplib.h](#) строка [1059](#)

6.36.6.35 `socket_options_`

`SocketOptions httpplib::Server::socket_options_ = default_socket_options` [private]

См. определение в файле [httpplib.h](#) строка [1155](#)

6.36.6.36 `svr_sock_`

`std::atomic<socket_t> httpplib::Server::svr_sock_ { (-1) }` [protected]

См. определение в файле [httpplib.h](#) строка [1055](#)

`01055 {INVALID_SOCKET};`

6.36.6.37 `tcp_nodelay_`

`bool httpplib::Server::tcp_nodelay_ = false` [private]

См. определение в файле [httpplib.h](#) строка [1153](#)

6.36.6.38 `write_timeout_sec_`

`time_t httpplib::Server::write_timeout_sec_ = 5` [protected]

См. определение в файле [httpplib.h](#) строка [1060](#)

6.36.6.39 `write_timeout_usec_`

`time_t httpplib::Server::write_timeout_usec_ = 0` [protected]

См. определение в файле [httpplib.h](#) строка [1061](#)

Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.37 Класс SHA256

Реализация криптографического хеш-функции SHA-256.

```
#include <SHA256.h>
```

Граф связей класса SHA256:



Открытые статические члены

- static std::string **hash** (const std::string &input)
Вычисляет SHA-256 хеш от входной строки.

6.37.1 Подробное описание

Реализация криптографического хеш-функции SHA-256.

Алгоритм SHA-256 преобразует произвольную входную строку в 256-битную (64 символа в hex) хеш-сумму. Используется, например, для хранения паролей и проверки целостности данных.

Алгоритм состоит из следующих шагов:

- Паддинг (дополнение входа до кратного 512 бит)
- Разбиение на 512-битные блоки
- Расширение блока до 64 слов по 32 бита
- 64 раунда с использованием побитовых и арифметических операций, включая , , ch, maj
- Финальное объединение состояний в хеш

См. определение в файле [SHA256.h](#) строка 17

6.37.2 Методы

6.37.2.1 hash()

```
std::string SHA256::hash (  
    const std::string & input) [static]
```

Вычисляет SHA-256 хеш от входной строки.

Аргументы

input	Строка произвольной длины.
-------	----------------------------

Возвращает

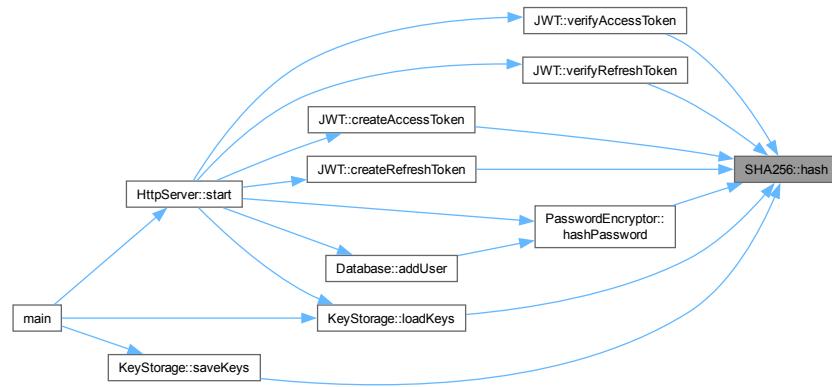
Хеш в шестнадцатеричном виде (64 символа).

См. определение в файле [SHA256.cpp](#) строка 120

```

00120      {
00121      std::cout << "[SHA256] --- Начало хеширования ---" << std::endl;
00122      std::cout << "[SHA256] Входная строка: " << input << std::endl;
00123
00124      std::array<uint32_t, 8> h = {
00125          0xa09e667, 0xb67ae85, 0x3c6ef372, 0xa54ff53a,
00126          0x510e527f, 0xb05688c, 0x1f83d9ab, 0x5be0cd19
00127      };
00128
00129      std::vector<uint8_t> data = pad(input);
00130      std::cout << "[SHA256] Размер после паддинга: " << data.size() << " байт" << std::endl;
00131
00132      for (size_t i = 0; i < data.size(); i += 64) {
00133          uint32_t w[64];
00134
00135          for (int j = 0; j < 16; ++j)
00136              w[j] = to_uint32(&data[i + j * 4]);
00137
00138          for (int j = 16; j < 64; ++j)
00139              w[j] = small_sigma1(w[j - 2]) + w[j - 7] +
00140                  small_sigma0(w[j - 15]) + w[j - 16];
00141
00142          uint32_t a = h[0], b = h[1], c = h[2], d = h[3];
00143          uint32_t e = h[4], f = h[5], g = h[6], h_val = h[7];
00144
00145          std::cout << "[SHA256] Обработка блока #" << i / 64 << std::endl;
00146
00147          for (int j = 0; j < 64; ++j) {
00148              uint32_t temp1 = h_val + big_sigma1(e) + ch(e, f, g) + k[j] + w[j];
00149              uint32_t temp2 = big_sigma0(a) + maj(a, b, c);
00150
00151              h_val = g;
00152              g = f;
00153              f = e;
00154              e = d + temp1;
00155              d = c;
00156              c = b;
00157              b = a;
00158              a = temp1 + temp2;
00159
00160              if (j < 4 || j > 59) { // лог только в начале и в конце
00161                  std::cout << "Round " << j << ": a=" << std::hex << a << " e=" << e << std::dec << std::endl;
00162              }
00163          }
00164
00165          h[0] += a; h[1] += b; h[2] += c; h[3] += d;
00166          h[4] += e; h[5] += f; h[6] += g; h[7] += h_val;
00167      }
00168
00169      std::ostringstream oss;
00170      for (auto val : h) {
00171          oss << std::hex << std::setw(8) << std::setfill('0') << val;
00172      }
00173
00174      std::string result = oss.str();
00175      std::cout << "[SHA256] Итоговый хеш: " << result << std::endl;
00176      std::cout << "[SHA256] --- Завершено ---" << std::endl;
00177
00178      return result;
00179 }
```

Граф вызова функций:



Объявления и описания членов классов находятся в файлах:

- [SHA256.h](#)
- [SHA256.cpp](#)

6.38 Структура `httpplib::ClientImpl::Socket`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::ClientImpl::Socket`:

<code>httpplib::ClientImpl</code>
<code>::Socket</code>
+ <code>sock</code>
+ <code>is_open()</code>

Открытые члены

- `bool is_open () const`

Открытые атрибуты

- `socket_t sock = (-1)`

6.38.1 Подробное описание

См. определение в файле [httpplib.h](#) строка [1502](#)

6.38.2 Методы

6.38.2.1 `is_open()`

```
bool httpplib::ClientImpl::Socket::is_open () const [inline]
```

См. определение в файле [httpplib.h](#) строка [1508](#)

```
01508 { return sock != INVALID_SOCKET; }
```

6.38.3 Данные класса

6.38.3.1 `sock`

```
socket_t httpplib::ClientImpl::Socket::sock = (-1)
```

См. определение в файле [httpplib.h](#) строка [1503](#)

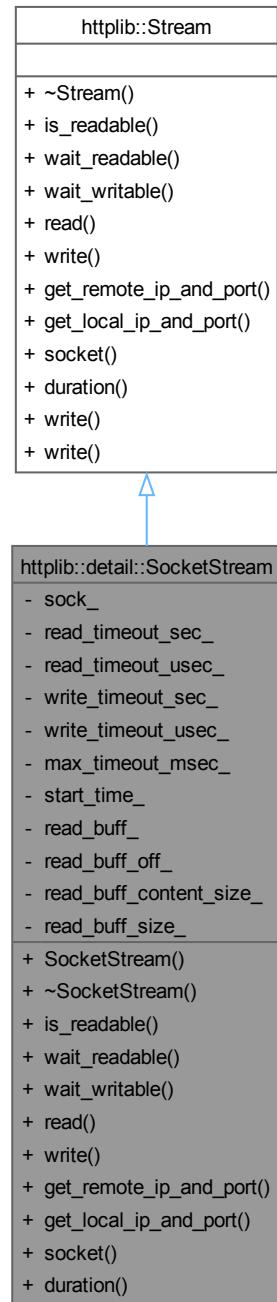
Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

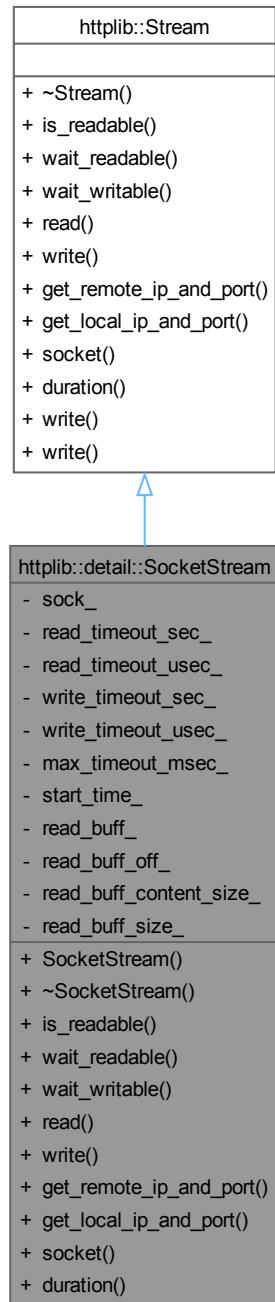
6.39 Класс `httpplib::detail::SocketStream`

```
#include <httpplib.h>
```

Граф наследования: httpplib::detail::SocketStream:



Граф связей класса `httpplib::detail::SocketStream`:



Открытые члены

- `SocketStream (socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, time_t max_timeout_msec=0, std::chrono::time_point< std::chrono::steady_clock > start_time=(std::chrono::steady_clock::time_point< ::min>()))`
- `~SocketStream () override`

- `bool is_readable () const override`
- `bool wait_readable () const override`
- `bool wait_writable () const override`
- `ssize_t read (char *ptr, size_t size) override`
- `ssize_t write (const char *ptr, size_t size) override`
- `void get_remote_ip_and_port (std::string &ip, int &port) const override`
- `void get_local_ip_and_port (std::string &ip, int &port) const override`
- `socket_t socket () const override`
- `time_t duration () const override`

Открытые члены унаследованные от `httpplib::Stream`

- `virtual ~Stream ()=default`
- `ssize_t write (const char *ptr)`
- `ssize_t write (const std::string &s)`

Закрытые данные

- `socket_t sock_`
- `time_t read_timeout_sec_`
- `time_t read_timeout_usec_`
- `time_t write_timeout_sec_`
- `time_t write_timeout_usec_`
- `time_t max_timeout_msec_`
- `const std::chrono::time_point<std::chrono::steady_clock> start_time_`
- `std::vector<char> read_buff_`
- `size_t read_buff_off_ = 0`
- `size_t read_buff_content_size_ = 0`

Закрытые статические данные

- `static const size_t read_buff_size_ = 1024l * 4`

6.39.1 Подробное описание

См. определение в файле `httpplib.h` строка [3345](#)

6.39.2 Конструктор(ы)

6.39.2.1 `SocketStream()`

```
httpplib::detail::SocketStream::SocketStream (
    socket_t sock,
    time_t read_timeout_sec,
    time_t read_timeout_usec,
    time_t write_timeout_sec,
    time_t write_timeout_usec,
    time_t max_timeout_msec = 0,
    std::chrono::time_point< std::chrono::steady_clock > start_time = (std::chrono::steady_clock::time_point::min)() ) [inline]
```

См. определение в файле `httpplib.h` строка 6065

```
06070 : sock_(sock), read_timeout_sec_(read_timeout_sec),
06071     read_timeout_usec_(read_timeout_usec),
06072     write_timeout_sec_(write_timeout_sec),
06073     write_timeout_usec_(write_timeout_usec),
06074     max_timeout_msec_(max_timeout_msec), start_time_(start_time),
06075     read_buff_(read_buff_size_, 0) {}
```

Граф вызовов:



Граф вызова функции:



6.39.2.2 `~SocketStream()`

```
httpplib::detail::SocketStream::~SocketStream () [inline], [override], [default]
```

6.39.3 Методы

6.39.3.1 `duration()`

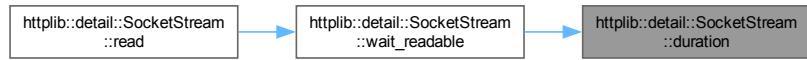
```
time_t httpplib::detail::SocketStream::duration () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6170

```
06170 {  
06171     return std::chrono::duration_cast<std::chrono::milliseconds>(  
06172         std::chrono::steady_clock::now() - start_time_)  
06173     .count();  
06174 }
```

Граф вызова функции:



6.39.3.2 `get_local_ip_and_port()`

```
void httpplib::detail::SocketStream::get_local_ip_and_port (  
    std::string & ip,  
    int & port) const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6163

```
06164 {  
06165     return detail::get_local_ip_and_port(sock_, ip, port);  
06166 }
```

Граф вызовов:



6.39.3.3 `get_remote_ip_and_port()`

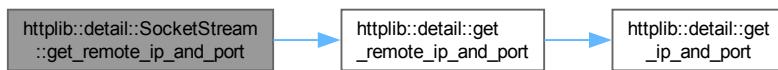
```
void httpplib::detail::SocketStream::get_remote_ip_and_port (
    std::string & ip,
    int & port) const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6158

```
06159     {
06160     return detail::get_remote_ip_and_port(sock_, ip, port);
06161 }
```

Граф вызовов:



6.39.3.4 `is_readable()`

```
bool httpplib::detail::SocketStream::is_readable () const [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6079

```
06079     {
06080     return read_buff_off_ < read_buff_content_size_;
06081 }
```

6.39.3.5 `read()`

```
ssize_t httpplib::detail::SocketStream::read (
    char * ptr,
    size_t size) [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6101

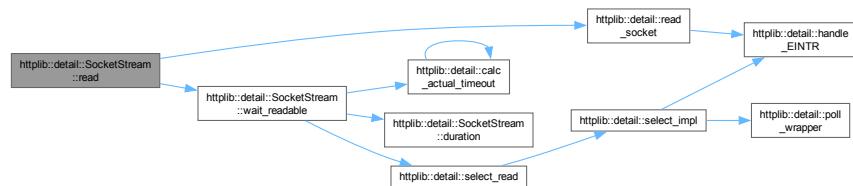
```
06101     {
06102 #ifdef _WIN32
06103     size =
06104         (std::min)(size, static_cast<size_t>((std::numeric_limits<int>::max)()));
06105 #else
06106     size = (std::min)(size,
06107                         static_cast<size_t>((std::numeric_limits<ssize_t>::max)()));
06108 #endif
06109
06110     if (read_buff_off_ < read_buff_content_size_) {
06111         auto remaining_size = read_buff_content_size_ - read_buff_off_;
06112         if (size <= remaining_size) {
06113             memcpy(ptr, read_buff_.data() + read_buff_off_, size);
06114             read_buff_off_ += size;
06115             return static_cast<ssize_t>(size);
06116         } else {
06117             memcpy(ptr, read_buff_.data() + read_buff_off_, remaining_size);
```

```

06118     read_buff_off_ += remaining_size;
06119     return static_cast<ssize_t>(remaining_size);
06120 }
06121 }
06122 if (!wait_readable()) { return -1; }
06124
06125 read_buff_off_ = 0;
06126 read_buff_content_size_ = 0;
06127
06128 if (size < read_buff_size_) {
06129     auto n = read_socket(sock_, read_buff_.data(), read_buff_size_,
06130                           CPPHTTPLIB_RECV_FLAGS);
06131     if (n <= 0) {
06132         return n;
06133     } else if (n <= static_cast<ssize_t>(size)) {
06134         memcpy(ptr, read_buff_.data(), static_cast<size_t>(n));
06135         return n;
06136     } else {
06137         memcpy(ptr, read_buff_.data(), size);
06138         read_buff_off_ = size;
06139         read_buff_content_size_ = static_cast<size_t>(n);
06140         return static_cast<ssize_t>(size);
06141     }
06142 } else {
06143     return read_socket(sock_, ptr, size, CPPHTTPLIB_RECV_FLAGS);
06144 }
06145 }

```

Граф вызовов:



6.39.3.6 socket()

`socket_t` `httpplib::detail::SocketStream::socket () const` [inline], [override], [virtual]

Замещает `httpplib::Stream`.

См. определение в файле `httpplib.h` строка 6168

```
06168 { return sock_; }
```

6.39.3.7 wait_readable()

`bool` `httpplib::detail::SocketStream::wait_readable () const` [inline], [override], [virtual]

Замещает `httpplib::Stream`.

См. определение в файле `httpplib.h` строка 6083

```

06083     if (max_timeout_msec_ <= 0) {
06084         return select_read(sock_, read_timeout_sec_, read_timeout_usec_) > 0;
06085     }
06086
06087     time_t read_timeout_sec;
06088     time_t read_timeout_usec;
06089     calc_actual_timeout(max_timeout_msec_, duration(), read_timeout_sec_,
06090                          read_timeout_usec_, read_timeout_sec, read_timeout_usec);
06091

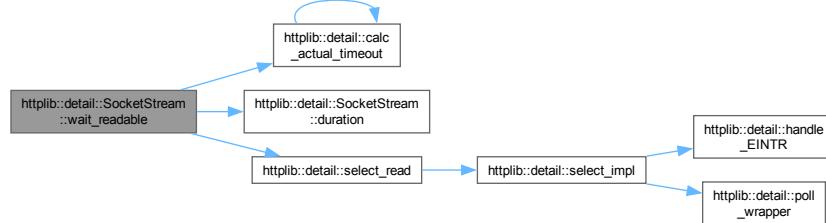
```

```

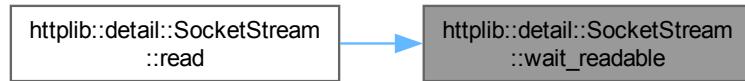
06092
06093     return select_read(sock_, read_timeout_sec, read_timeout_usec) > 0;
06094 }

```

Граф вызовов:



Граф вызова функции:



6.39.3.8 `wait_writable()`

`bool httpplib::detail::SocketStream::wait_writable () const [inline], [override], [virtual]`

Заменяет [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6096

```

06096
06097     return select_write(sock_, write_timeout_sec_, write_timeout_usec_) > 0 &&
06098         is_socket_alive(sock_);
06099 }

```

Граф вызовов:



Граф вызова функции:



6.39.3.9 write()

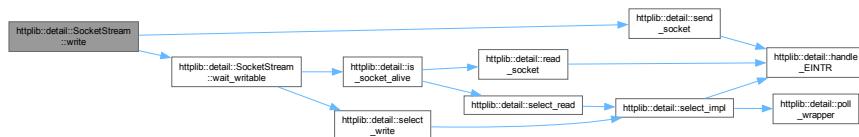
```
ssize_t httpplib::detail::SocketStream::write (
    const char * ptr,
    size_t size) [inline], [override], [virtual]
```

Замещает [httpplib::Stream](#).

См. определение в файле [httpplib.h](#) строка 6147

```
06147
06148     if (!wait_writable()) { return -1; }
06149
06150 #if defined(_WIN32) && !defined(_WIN64)
06151     size =
06152         (std::min)(size, static_cast<size_t>((std::numeric_limits<int>::max())));
06153 #endif
06154
06155     return send_socket(sock_, ptr, size, CPPHTTPPLIB_SEND_FLAGS);
06156 }
```

Граф вызовов:



6.39.4 Данные класса

6.39.4.1 max_timeout_msec_

```
time_t httpplib::detail::SocketStream::max_timeout_msec_ [private]
```

См. определение в файле [httpplib.h](#) строка 3370

6.39.4.2 read_buff_

```
std::vector<char> httpplib::detail::SocketStream::read_buff_ [private]
```

См. определение в файле [httpplib.h](#) строка 3373

6.39.4.3 `read_buff_content_size_`

`size_t httpplib::detail::SocketStream::read_buff_content_size_ = 0` [private]

См. определение в файле [httpplib.h](#) строка [3375](#)

6.39.4.4 `read_buff_off_`

`size_t httpplib::detail::SocketStream::read_buff_off_ = 0` [private]

См. определение в файле [httpplib.h](#) строка [3374](#)

6.39.4.5 `read_buff_size_`

`const size_t httpplib::detail::SocketStream::read_buff_size_ = 1024l * 4` [static], [private]

См. определение в файле [httpplib.h](#) строка [3377](#)

6.39.4.6 `read_timeout_sec_`

`time_t httpplib::detail::SocketStream::read_timeout_sec_` [private]

См. определение в файле [httpplib.h](#) строка [3366](#)

6.39.4.7 `read_timeout_usec_`

`time_t httpplib::detail::SocketStream::read_timeout_usec_` [private]

См. определение в файле [httpplib.h](#) строка [3367](#)

6.39.4.8 `sock_`

`socket_t httpplib::detail::SocketStream::sock_` [private]

См. определение в файле [httpplib.h](#) строка [3365](#)

6.39.4.9 `start_time_`

`const std::chrono::time_point<std::chrono::steady_clock> httpplib::detail::SocketStream::start_time_` [private]

См. определение в файле [httpplib.h](#) строка [3371](#)

6.39.4.10 `write_timeout_sec_`

`time_t httpplib::detail::SocketStream::write_timeout_sec_` [private]

См. определение в файле [httpplib.h](#) строка [3368](#)

6.39.4.11 `write_timeout_usec_`

```
time_t httpplib::detail::SocketStream::write_timeout_usec_ [private]
```

См. определение в файле [httpplib.h](#) строка 3369

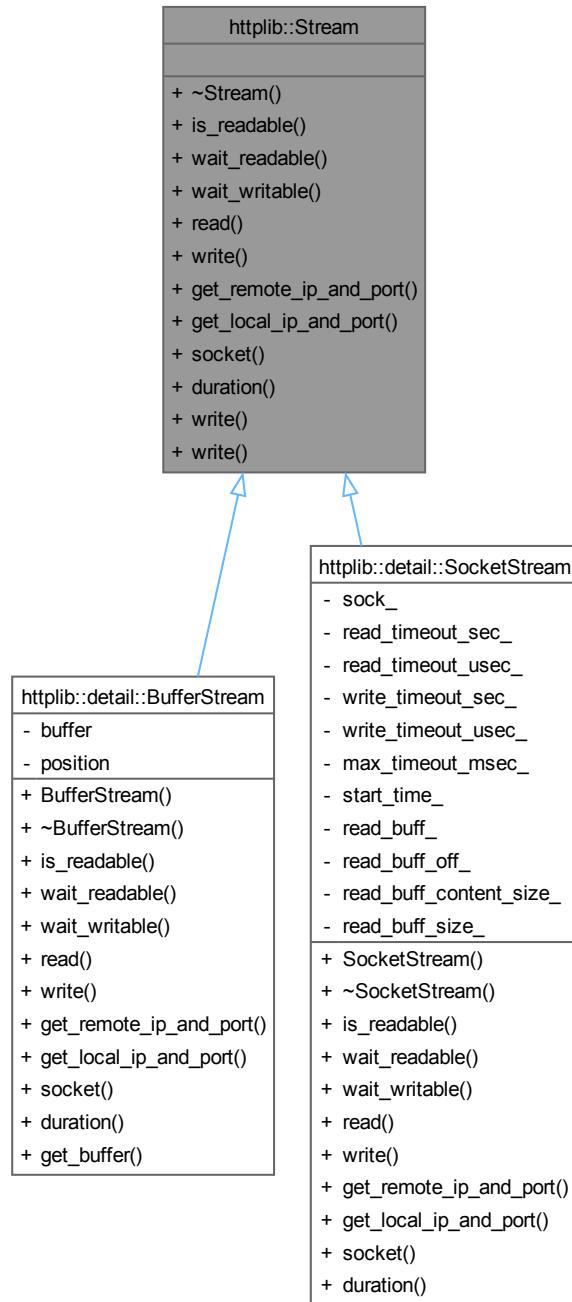
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

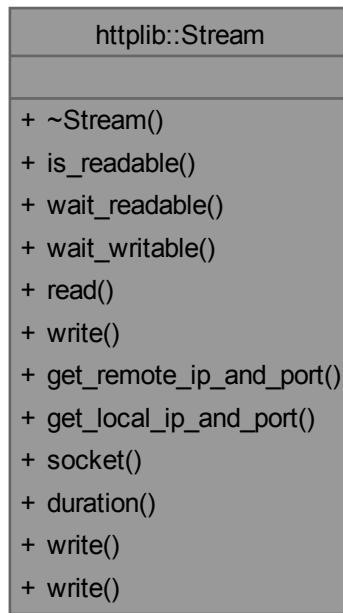
6.40 Класс `httpplib::Stream`

```
#include <httpplib.h>
```

Граф наследования: httplib::Stream:



Граф связей класса `httpplib::Stream`:



Открытые члены

- `virtual ~Stream ()=default`
- `virtual bool is_readable () const =0`
- `virtual bool wait_readable () const =0`
- `virtual bool wait_writable () const =0`
- `virtual ssize_t read (char *ptr, size_t size)=0`
- `virtual ssize_t write (const char *ptr, size_t size)=0`
- `virtual void get_remote_ip_and_port (std::string &ip, int &port) const =0`
- `virtual void get_local_ip_and_port (std::string &ip, int &port) const =0`
- `virtual socket_t socket () const =0`
- `virtual time_t duration () const =0`
- `ssize_t write (const char *ptr)`
- `ssize_t write (const std::string &s)`

6.40.1 Подробное описание

См. определение в файле [httpplib.h](#) строка [742](#)

6.40.2 Конструктор(ы)

6.40.2.1 `~Stream()`

`virtual httpplib::Stream::~Stream () [virtual], [default]`

6.40.3 Методы

6.40.3.1 `duration()`

```
virtual time_t httpplib::Stream::duration () const [pure virtual]
```

Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

6.40.3.2 `get_local_ip_and_port()`

```
virtual void httpplib::Stream::get_local_ip_and_port (
    std::string & ip,
    int & port) const [pure virtual]
```

Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

6.40.3.3 `get_remote_ip_and_port()`

```
virtual void httpplib::Stream::get_remote_ip_and_port (
    std::string & ip,
    int & port) const [pure virtual]
```

Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

6.40.3.4 `is_readable()`

```
virtual bool httpplib::Stream::is_readable () const [pure virtual]
```

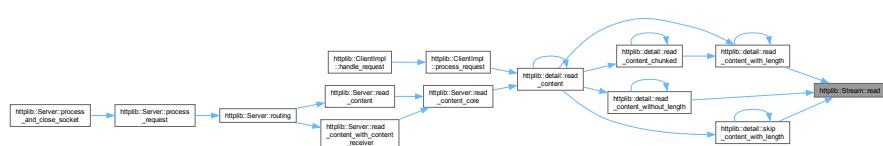
Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

6.40.3.5 `read()`

```
virtual ssize_t httpplib::Stream::read (
    char * ptr,
    size_t size) [pure virtual]
```

Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

Граф вызова функции:



6.40.3.6 `socket()`

```
virtual socket_t httpplib::Stream::socket () const [pure virtual]
```

Замещается в `httpplib::detail::BufferStream` и `httpplib::detail::SocketStream`.

Граф вызова функции:



6.40.3.7 `wait_readable()`

```
virtual bool httpplib::Stream::wait_readable () const [pure virtual]
```

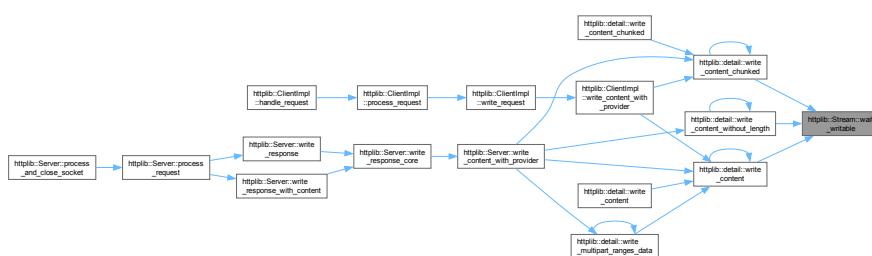
Замещается в `httpplib::detail::BufferStream` и `httpplib::detail::SocketStream`.

6.40.3.8 `wait_writable()`

```
virtual bool httpplib::Stream::wait_writable () const [pure virtual]
```

Замещается в `httpplib::detail::BufferStream` и `httpplib::detail::SocketStream`.

Граф вызова функции:



6.40.3.9 `write()` [1/3]

```
ssize_t httpplib::Stream::write (
    const char * ptr) [inline]
```

См. определение в файле [httpplib.h](#) строка 6039

```
06039 {
```

```
06040     return write(ptr, strlen(ptr));
```

```
06041 }
```

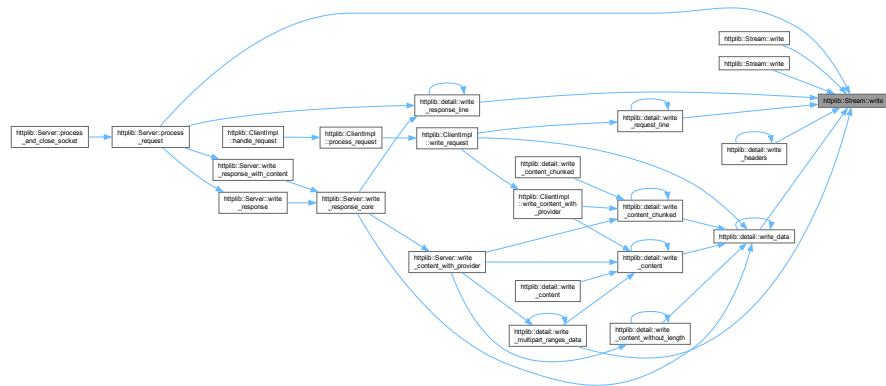
Граф вызовов:

6.40.3.10 `write()` [2/3]

```
virtual ssize_t httpplib::Stream::write (
    const char * ptr,
    size_t size) [pure virtual]
```

Замещается в [httpplib::detail::BufferStream](#) и [httpplib::detail::SocketStream](#).

Граф вызова функции:



6.40.3.11 `write()` [3/3]

```
ssize_t httpplib::Stream::write (
    const std::string & s) [inline]
```

См. определение в файле [httpplib.h](#) строка [6043](#)

```
06043 {  
06044     return write(s.data(), s.size());  
06045 }
```

Граф вызовов:



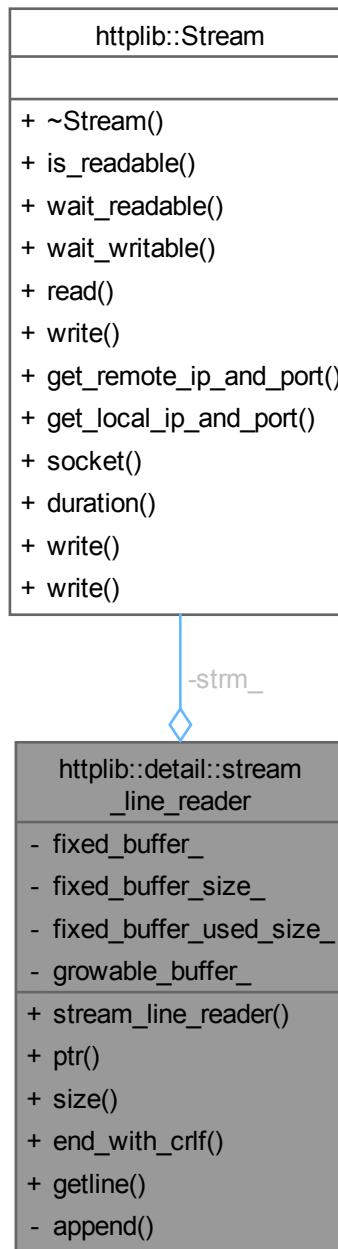
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

6.41 Класс `httpplib::detail::stream_line_reader`

```
#include <httpplib.h>
```

Граф связей класса `httpplib::detail::stream_line_reader`:



Открытые члены

- `stream_line_reader (Stream &strm, char *fixed_buffer, size_t fixed_buffer_size)`
- `const char * ptr () const`
- `size_t size () const`
- `bool end_with_crlf () const`
- `bool getline ()`

Закрытые члены

- `void append (char c)`

Закрытые данные

- `Stream & strm_`
- `char * fixed_buffer_`
- `const size_t fixed_buffer_size_`
- `size_t fixed_buffer_used_size_ = 0`
- `std::string growable_buffer_`

6.41.1 Подробное описание

См. определение в файле `httpplib.h` строка 2598

6.41.2 Конструктор(ы)

6.41.2.1 `stream_line_reader()`

```
httpplib::detail::stream_line_reader::stream_line_reader (
    Stream & strm,
    char * fixed_buffer,
    size_t fixed_buffer_size) [inline]
```

См. определение в файле `httpplib.h` строка 3035

```
03037     : strm_(strm), fixed_buffer_(fixed_buffer),
03038         fixed_buffer_size_(fixed_buffer_size) {}
```

6.41.3 Методы

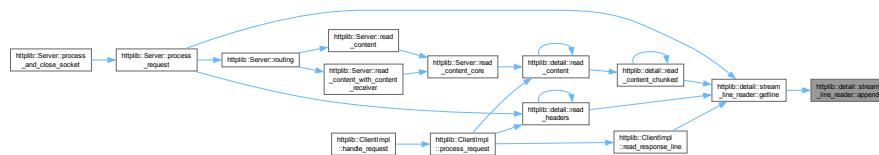
6.41.3.1 `append()`

```
void httpplib::detail::stream_line_reader::append (
    char c) [inline], [private]
```

См. определение в файле `httpplib.h` строка 3096

```
03096     {
03097     if (fixed_buffer_used_size_ < fixed_buffer_size_ - 1) {
03098         fixed_buffer_[fixed_buffer_used_size_ + 1] = c;
03099         fixed_buffer_[fixed_buffer_used_size_] = '\0';
03100     } else {
03101         if (growable_buffer_.empty()) {
03102             assert(fixed_buffer_[fixed_buffer_used_size_] == '\0');
03103             growable_buffer_.assign(fixed_buffer_, fixed_buffer_used_size_);
03104         }
03105         growable_buffer_ += c;
03106     }
03107 }
```

Граф вызова функции:



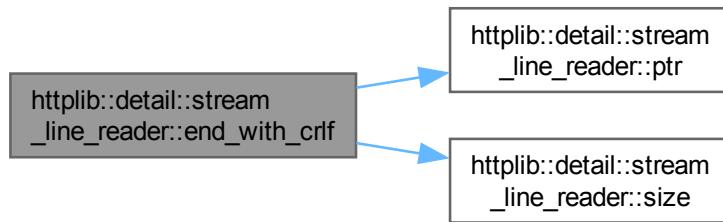
6.41.3.2 end_with_crlf()

```
bool httpplib::detail::stream_line_reader::end_with_crlf () const [inline]
```

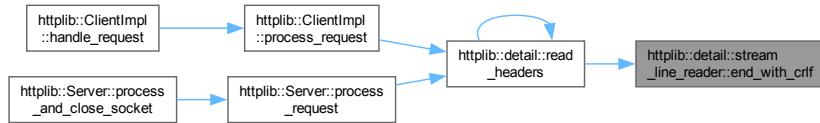
См. определение в файле [httpplib.h](#) строка 3056

```
03056     {
03057     auto end = ptr() + size();
03058     return size() >= 2 && end[-2] == '\r' && end[-1] == '\n';
03059 }
```

Граф вызовов:



Граф вызова функции:



6.41.3.3 getline()

```
bool httpplib::detail::stream_line_reader::getline () [inline]
```

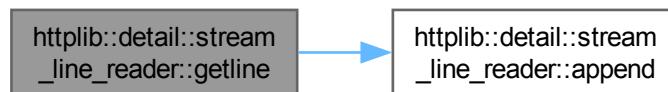
См. определение в файле [httpplib.h](#) строка 3061

```
03061     {
03062     fixed_buffer_used_size_ = 0;
03063     growable_buffer_.clear();
03064
03065 #ifndef CPPHTTPPLIB_ALLOW_LF_AS_LINE_TERMINATOR
03066     char prev_byte = 0;
03067 #endif
03068
03069     for (size_t i = 0;; i++) {
03070         char byte;
03071         auto n = strm_.read(&byte, 1);
03072
03073         if (n < 0) {
03074             return false;
03075         } else if (n == 0) {
03076             if (i == 0) {
03077                 return false;
03078             }
03079         }
03080     }
03081 }
```

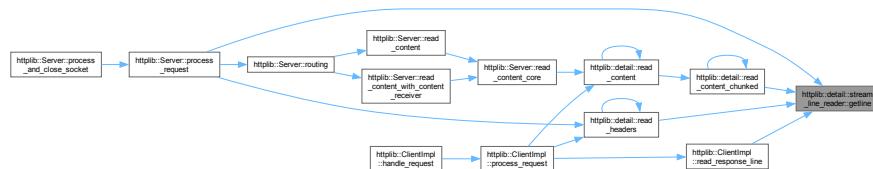
```

03078     } else {
03079         break;
03080     }
03081 }
03082
03083     append(byte);
03084
03085 #ifdef CPPHTTPPLIB_ALLOW_LF_AS_LINE_TERMINATOR
03086     if (byte == '\n') { break; }
03087 #else
03088     if (prev_byte == '\r' && byte == '\n') { break; }
03089     prev_byte = byte;
03090 #endif
03091 }
03092
03093 return true;
03094 }
```

Граф вызовов:



Граф вызова функции:



6.41.3.4 `ptr()`

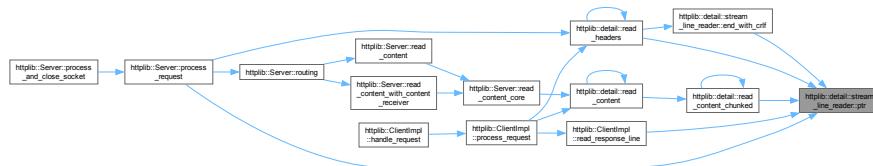
`const char * httpplib::detail::stream_line_reader::ptr () const [inline]`

См. определение в файле `httpplib.h` строка 3040

```

03040
03041     if (growable_buffer_.empty()) {
03042         return fixed_buffer_;
03043     } else {
03044         return growable_buffer_.data();
03045     }
03046 }
```

Граф вызова функции:



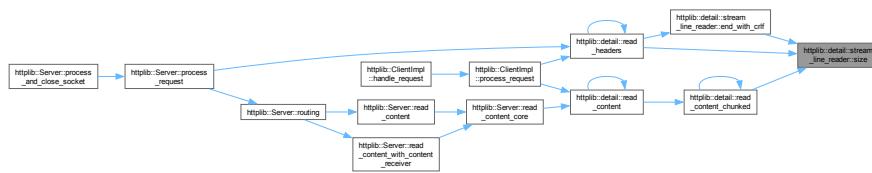
6.41.3.5 `size()`

`size_t httpplib::detail::stream_line_reader::size () const [inline]`

См. определение в файле [httpplib.h](#) строка 3048

```
03048 if (growable_buffer_.empty()) {
03049     return fixed_buffer_used_size_;
03050 } else {
03051     return growable_buffer_.size();
03052 }
03053 }
```

Граф вызова функции:



6.41.4 Данные класса

6.41.4.1 `fixed_buffer_`

`char* httpplib::detail::stream_line_reader::fixed_buffer_ [private]`

См. определение в файле [httpplib.h](#) строка 2611

6.41.4.2 `fixed_buffer_size_`

`const size_t httpplib::detail::stream_line_reader::fixed_buffer_size_ [private]`

См. определение в файле [httpplib.h](#) строка 2612

6.41.4.3 `fixed_buffer_used_size_`

`size_t httpplib::detail::stream_line_reader::fixed_buffer_used_size_ = 0 [private]`

См. определение в файле [httpplib.h](#) строка 2613

6.41.4.4 `growable_buffer_`

`std::string httpplib::detail::stream_line_reader::growable_buffer_ [private]`

См. определение в файле [httpplib.h](#) строка 2614

6.41.4.5 `strm_`

```
Stream& httpplib::detail::stream_line_reader::strm_ [private]
```

См. определение в файле [httpplib.h](#) строка 2610

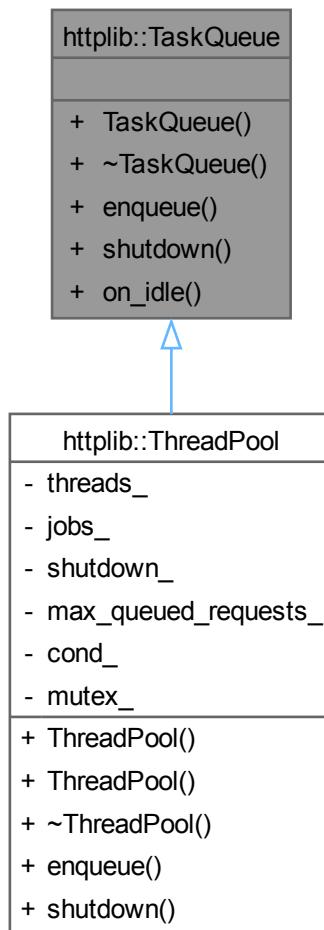
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

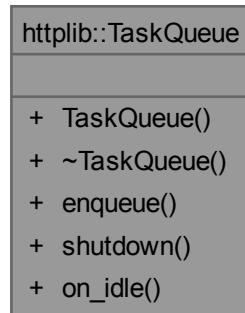
6.42 Класс `httpplib::TaskQueue`

```
#include <httpplib.h>
```

Граф наследования:`httpplib::TaskQueue`:



Граф связей класса `httpplib::TaskQueue`:



Открытые члены

- `TaskQueue ()=default`
- `virtual ~TaskQueue ()=default`
- `virtual bool enqueue (std::function< void()> fn)=0`
- `virtual void shutdown ()=0`
- `virtual void on_idle ()`

6.42.1 Подробное описание

См. определение в файле [httpplib.h](#) строка 762

6.42.2 Конструктор(ы)

6.42.2.1 `TaskQueue()`

`httpplib::TaskQueue::TaskQueue () [default]`

6.42.2.2 `~TaskQueue()`

`virtual httpplib::TaskQueue::~TaskQueue () [virtual], [default]`

6.42.3 Методы

6.42.3.1 `enqueue()`

`virtual bool httpplib::TaskQueue::enqueue (`
`std::function< void()> fn) [pure virtual]`

Замещается в [httpplib::ThreadPool](#).

6.42.3.2 on_idle()

virtual void httpplib::TaskQueue::on_idle () [inline], [virtual]

См. определение в файле [httpplib.h](#) строка 770

00770 {}

6.42.3.3 shutdown()

virtual void httpplib::TaskQueue::shutdown () [pure virtual]

Замещается в [httpplib::ThreadPool](#).

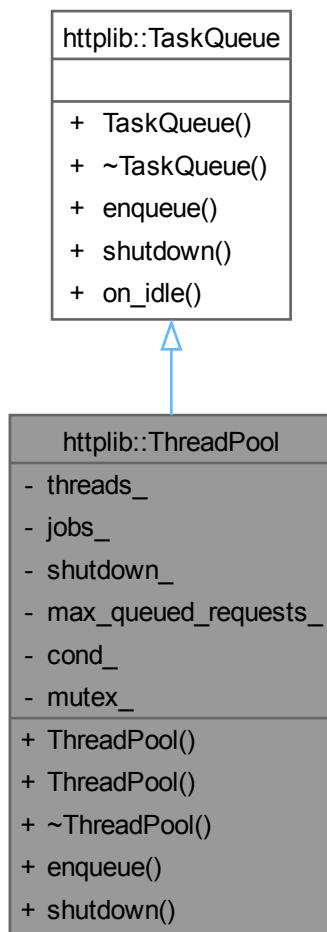
Объявления и описания членов класса находятся в файле:

- [httpplib.h](#)

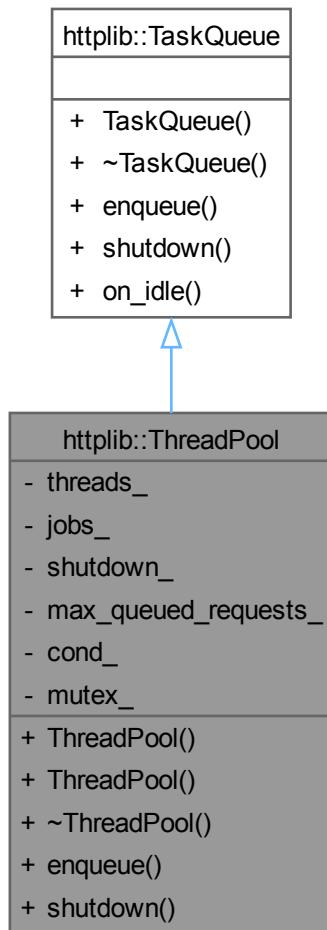
6.43 Класс httpplib::ThreadPool

#include <httpplib.h>

Граф наследования: httpplib::ThreadPool:



Граф связей класса httpplib::ThreadPool:



Классы

- struct `worker`

Открытые члены

- `ThreadPool` (`size_t n, size_t mqr=0`)
- `ThreadPool` (`const ThreadPool &`)=`delete`
- `~ThreadPool` () `override=default`
- `bool enqueue` (`std::function< void()> fn`) `override`
- `void shutdown` () `override`

Открытые члены унаследованные от `httpplib::TaskQueue`

- `TaskQueue` ()=`default`
- `virtual ~TaskQueue` ()=`default`
- `virtual void on_idle` ()

Закрытые данные

- `std::vector< std::thread > threads_`
- `std::list< std::function< void()> > jobs_`
- `bool shutdown_`
- `size_t max_queued_requests_ = 0`
- `std::condition_variable cond_`
- `std::mutex mutex_`

Друзья

- `struct worker`

6.43.1 Подробное описание

См. определение в файле `httpplib.h` строка [773](#)

6.43.2 Конструктор(ы)

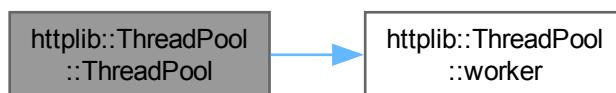
6.43.2.1 `ThreadPool()` [1/2]

```
httpplib::ThreadPool::ThreadPool (
    size_t n,
    size_t mqr = 0)  [inline], [explicit]
```

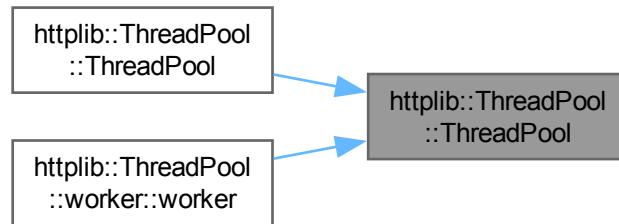
См. определение в файле `httpplib.h` строка [775](#)

```
00776     : shutdown_(false), max_queued_requests_(mqr) {
00777     while (n) {
00778         threads_.emplace_back(worker(*this));
00779         n--;
00780     }
00781 }
```

Граф вызовов:



Граф вызова функции:



6.43.2.2 ThreadPool() [2/2]

```
http://www.libhttp.org/html/httplib/ThreadPool.html#L100
httplib::ThreadPool::ThreadPool (
    const ThreadPool & ) [delete]
```

Граф вызовов:



6.43.2.3 ~ThreadPool()

```
http://www.libhttp.org/html/httplib/ThreadPool.html#L100
http://www.libhttp.org/html/httplib/ThreadPool.html#L100 ~ThreadPool () [override], [default]
```

6.43.3 Методы

6.43.3.1 enqueue()

```
bool http://www.libhttp.org/html/httplib/ThreadPool.html#L100
http://www.libhttp.org/html/httplib/ThreadPool.html#L100 enqueue (
    std::function< void()> fn) [inline], [override], [virtual]
```

Замещает [http://www.libhttp.org/html/httplib/TaskQueue.html#L100](#).

См. определение в файле [http://www.libhttp.org/html/httplib.h](#) строка 786

```

00786
00787 {
00788     std::unique_lock<std::mutex> lock(mutex_);
00789     if (max_queued_requests_ > 0 && jobs_.size() >= max_queued_requests_) {
00790         return false;
00791     }
00792     jobs_.push_back(std::move(fn));
00793 }
00794
00795 cond_.notify_one();
00796 return true;
00797 }
```

6.43.3.2 `shutdown()`

`void httpplib::ThreadPool::shutdown () [inline], [override], [virtual]`

Замещает [httpplib::TaskQueue](#).

См. определение в файле [httpplib.h](#) строка 799

```
00799     {  
00800     // Stop all worker threads...  
00801     {  
00802         std::unique_lock<std::mutex> lock(mutex_);  
00803         shutdown_ = true;  
00804     }  
00805     cond_.notify_all();  
00806     // Join...  
00807     for (auto &t : threads_) {  
00808         t.join();  
00809     }  
00810 }
```

6.43.4 Друзья класса и относящимся к классу обозначения

6.43.4.1 `worker`

`friend struct worker [friend]`

См. определение в файле [httpplib.h](#) строка 845

6.43.5 Данные класса

6.43.5.1 `cond_`

`std::condition_variable httpplib::ThreadPool::cond_ [private]`

См. определение в файле [httpplib.h](#) строка 853

6.43.5.2 `jobs_`

`std::list<std::function<void()> > httpplib::ThreadPool::jobs_ [private]`

См. определение в файле [httpplib.h](#) строка 848

6.43.5.3 `max_queued_requests_`

`size_t httpplib::ThreadPool::max_queued_requests_ = 0 [private]`

См. определение в файле [httpplib.h](#) строка 851

6.43.5.4 `mutex_`

`std::mutex httpplib::ThreadPool::mutex_ [private]`

См. определение в файле [httpplib.h](#) строка 854

6.43.5.5 shutdown_

```
bool httplib::ThreadPool::shutdown_ [private]
```

См. определение в файле [httplib.h](#) строка 850

6.43.5.6 threads_

```
std::vector<std::thread> httplib::ThreadPool::threads_ [private]
```

См. определение в файле [httplib.h](#) строка 847

Объявления и описания членов класса находятся в файле:

- [httplib.h](#)

6.44 Структура User

Структура, представляющая пользователя из базы данных.

```
#include <Database.h>
```

Граф связей класса User:



Открытые атрибуты

- int **id**

Уникальный идентификатор пользователя в таблице.

- std::string **username**

Имя пользователя.

- std::string **password**

Хеш пароля (уже захеширован).

6.44.1 Подробное описание

Структура, представляющая пользователя из базы данных.

См. определение в файле [Database.h](#) строка [8](#)

6.44.2 Данные класса

6.44.2.1 id

int User::id

Уникальный идентификатор пользователя в таблице.

См. определение в файле [Database.h](#) строка [9](#)

6.44.2.2 password

std::string User::password

Хеш пароля (уже захеширован).

См. определение в файле [Database.h](#) строка [11](#)

6.44.2.3 username

std::string User::username

Имя пользователя.

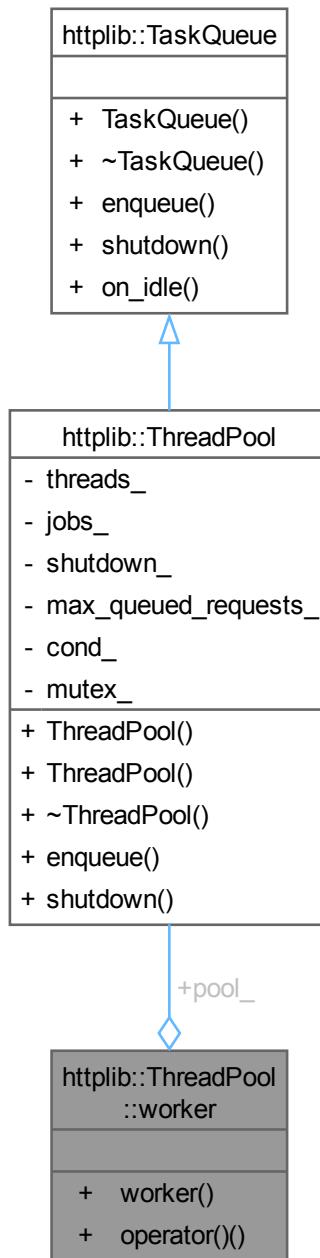
См. определение в файле [Database.h](#) строка [10](#)

Объявления и описания членов структуры находятся в файле:

- [Database.h](#)

6.45 Структура `httpplib::ThreadPool::worker`

Граф связей класса `httpplib::ThreadPool::worker`:



Открытые члены

- `worker (ThreadPool &pool)`
- `void operator() ()`

Открытые атрибуты

- `ThreadPool & pool_`

6.45.1 Подробное описание

См. определение в файле `httpplib.h` строка [815](#)

6.45.2 Конструктор(ы)

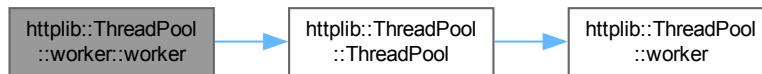
6.45.2.1 `worker()`

```
httpplib::ThreadPool::worker::worker (
    ThreadPool & pool) [inline], [explicit]
```

См. определение в файле `httpplib.h` строка [816](#)

```
00816 : pool_(pool) {}
```

Граф вызовов:



6.45.3 Методы

6.45.3.1 `operator()()`

```
void httpplib::ThreadPool::worker::operator() () [inline]
```

См. определение в файле `httpplib.h` строка [818](#)

```
00818     {
00819         for (;;) {
00820             std::function<void()> fn;
00821             {
00822                 std::unique_lock<std::mutex> lock(pool_.mutex_);
00823 
00824                 pool_.cond_.wait(
00825                     lock, [&] { return !pool_.jobs_.empty() || pool_.shutdown_; });
00826 
00827                 if (pool_.shutdown_ && pool_.jobs_.empty()) { break; }
00828 
00829                 fn = pool_.jobs_.front();
00830                 pool_.jobs_.pop_front();
00831             }
00832 
00833             assert(true == static_cast<bool>(fn));
00834             fn();
00835         }
00836 
00837 #if defined(CPPHTTPPLIB_OPENSSL_SUPPORT) && !defined(OPENSSL_IS_BORINGSSL) && \
00838     !defined(LIBRESSL_VERSION_NUMBER)
00839     OpenSSL_thread_stop();
00840 #endif
00841 }
```

6.45.4 Данные класса

6.45.4.1 `pool_`

`ThreadPool& httpplib::ThreadPool::worker::pool_`

См. определение в файле [httpplib.h](#) строка 843

Объявления и описания членов структуры находятся в файле:

- [httpplib.h](#)

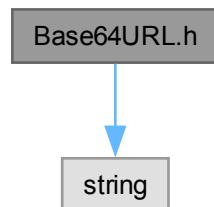
Глава 7

Файлы

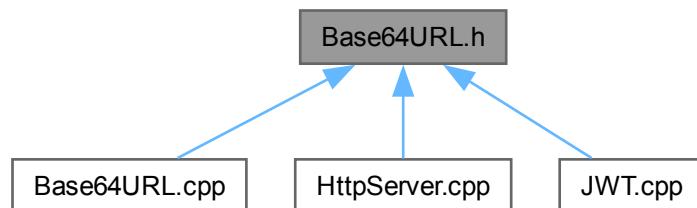
7.1 Файл Base64URL.h

```
#include <string>
```

Граф включаемых заголовочных файлов для Base64URL.h:



Граф файлов, в которые включается этот файл:



Классы

- class **Base64URL**

Класс для кодирования и декодирования строк в формате [Base64URL](#).

7.2 Base64URL.h

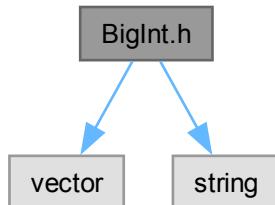
[См. документацию.](#)

```
00001 #pragma once
00002 #include <string>
00003
00017 class Base64URL {
00018 public:
00034     static std::string encode(const std::string& input);
00035
00048     static std::string decode(const std::string& input);
00049 };
```

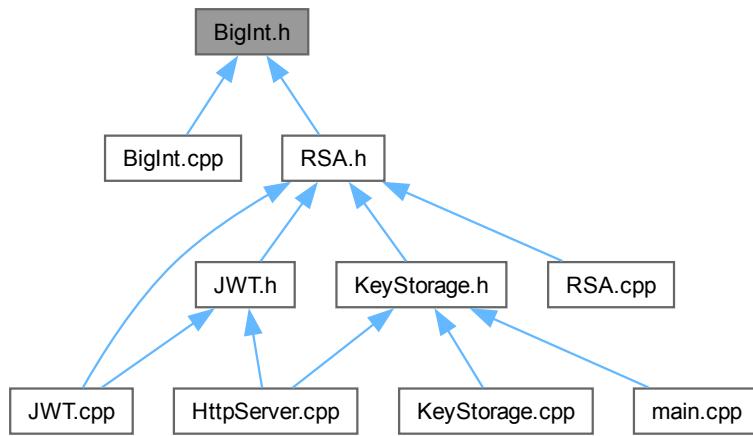
7.3 Файл BigInt.h

```
#include <vector>
#include <string>
```

Граф включаемых заголовочных файлов для BigInt.h:



Граф файлов, в которые включается этот файл:



Классы

- class **BigInt**

Класс для работы с большими целыми числами произвольной длины (Big Integer).

7.4 BigInt.h

[См. документацию.](#)

```

00001 #pragma once
00002 #include <vector>
00003 #include <string>
00004
00012 class BigInt {
00013 public:
00017   BigInt();
00018
00025   BigInt(int value);
00026
00033   BigInt(const std::string& str);
00034
00042   BigInt(const std::string& str, int base);
00043
00047   std::string toString() const;
00048
00055   std::string toString(int base) const;
00056
00057 // === Арифметические операции ===
00058
00067   BigInt operator+(const BigInt& other) const;
00068
00077   BigInt operator-(const BigInt& other) const;
00078
00087   BigInt operator*(const BigInt& other) const;
00088
00098   BigInt operator/(const BigInt& other) const;
00099
00107   BigInt operator%(const BigInt& other) const;
00108
00122   static BigInt modPow(BigInt base, BigInt exp, const BigInt& mod);
00123
00133   static BigInt gcd(BigInt a, BigInt b);
00134
00135 // === Операторы сравнения ===
  
```

```

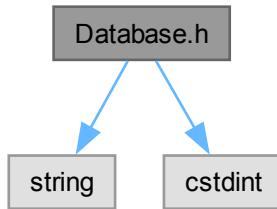
00136
00137     bool operator<(const BigInt& other) const;
00138     bool operator>(const BigInt& other) const;
00139     bool operator==(const BigInt& other) const;
00140     bool operator!=(const BigInt& other) const;
00141     bool operator<=(const BigInt& other) const;
00142     bool operator>=(const BigInt& other) const;
00143
00144     BigInt operator-() const;
00145
00146     bool isZero() const;
00147
00148     bool isNegative() const;
00149
00150 private:
00151     std::vector<int> digits;
00152     bool negative = false;
00153
00154     void trim();
00155
00156     static int compareAbs(const BigInt& a, const BigInt& b);
00157
00158 };

```

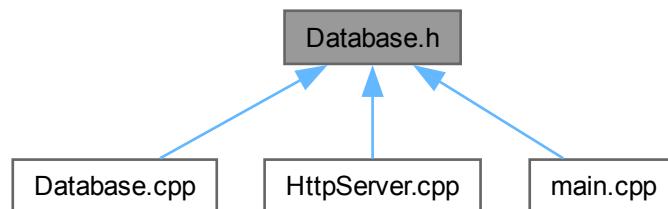
7.5 Файл Database.h

```
#include <string>
#include <cstdint>
```

Граф включаемых заголовочных файлов для Database.h:



Граф файлов, в которые включается этот файл:



Классы

- struct **User**
Структура, представляющая пользователя из базы данных.
- class **Database**
Класс-обёртка для работы с SQLite-базой данных.

7.6 Database.h

[См. документацию.](#)

```

00001 #pragma once
00002 #include <string>
00003 #include <cstdint>
00004
00008 struct User {
00009     int id;
0010     std::string username;
0011     std::string password;
0012 };
0013
0020 class Database {
0021 public:
0032     static bool init(const std::string& db_path);
0033
0043     static bool addUser(const std::string& username, const std::string& password);
0044
0054     static bool getUser(const std::string& username, User& user_out);
0055
0056 // ===== Методы для работы с blacklist токенов =====
0057
0067     static bool blacklistToken(const std::string& token, uint64_t expires_at);
0068
0077     static bool isTokenBlacklisted(const std::string& token);
0078
0086     static bool cleanupBlacklist();
0087 };

```

7.7 Файл httpplib.h

```

#include <arpa/inet.h>
#include <ifaddrs.h>
#include <net/if.h>
#include <netdb.h>
#include <netinet/in.h>
#include <csignal>
#include <netinet/tcp.h>
#include <poll.h>
#include <pthread.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <algorithm>
#include <array>
#include <atomic>
#include <cassert>
#include <cctype>
#include <climits>
#include <condition_variable>
#include <cstring>
#include <errno.h>

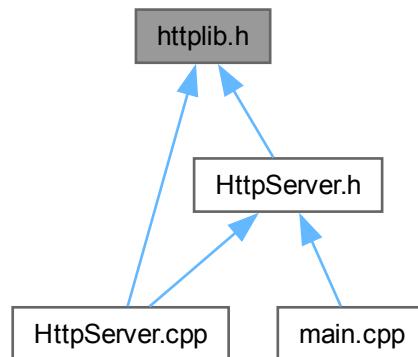
```

```
#include <exception>
#include <fcntl.h>
#include <functional>
#include <iomanip>
#include <iostream>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <random>
#include <regex>
#include <set>
#include <sstream>
#include <string>
#include <sys/stat.h>
#include <thread>
#include <unordered_map>
#include <unordered_set>
#include <utility>
```

Граф включаемых заголовочных файлов для `httpplib.h`:



Граф файлов, в которые включается этот файл:



Классы

- struct `httpplib::detail::case_ignore::equal_to`
- struct `httpplib::detail::case_ignore::hash`
- struct `httpplib::detail::scope_exit`
- struct `httpplib::MultipartFormData`
- class `httpplib::DataSink`
- class `httpplib::DataSink::data_sink_streambuf`
- struct `httpplib::MultipartFormDataProvider`

- class `httpplib::ContentReader`
- struct `httpplib::Request`
- struct `httpplib::Response`
- class `httpplib::Stream`
- class `httpplib::TaskQueue`
- class `httpplib::ThreadPool`
- struct `httpplib::ThreadPool::worker`
- class `httpplib::detail::MatcherBase`
- class `httpplib::detail::PathParamsMatcher`
- class `httpplib::detail::RegexMatcher`
- class `httpplib::Server`
- struct `httpplib::Server::MountPointEntry`
- class `httpplib::Result`
- class `httpplib::ClientImpl`
- struct `httpplib::ClientImpl::Socket`
- class `httpplib::Client`
- struct `httpplib::detail::FileStat`
- class `httpplib::detail::BufferStream`
- class `httpplib::detail::compressor`
- class `httpplib::detail::decompressor`
- class `httpplib::detail::nocompressor`
- class `httpplib::detail::stream_line_reader`
- class `httpplib::detail::mmap`
- class `httpplib::detail::SocketStream`
- class `httpplib::detail::MultipartFormDataParser`
- class `httpplib::detail::ContentProviderAdapter`

Пространства имен

- namespace `httpplib`
- namespace `httpplib::detail`
- namespace `httpplib::detail::case_ignore`
- namespace `httpplib::detail::fields`
- namespace `httpplib::detail::udl`

Макросы

- `#define CPPHTTPPLIB_VERSION "0.20.0"`
- `#define CPPHTTPPLIB_KEEPALIVE_TIMEOUT_SECOND 5`
- `#define CPPHTTPPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND 10000`
- `#define CPPHTTPPLIB_KEEPALIVE_MAX_COUNT 100`
- `#define CPPHTTPPLIB_CONNECTION_TIMEOUT_SECOND 300`
- `#define CPPHTTPPLIB_CONNECTION_TIMEOUT_USECOND 0`
- `#define CPPHTTPPLIB_SERVER_READ_TIMEOUT_SECOND 5`
- `#define CPPHTTPPLIB_SERVER_READ_TIMEOUT_USECOND 0`
- `#define CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_SECOND 5`
- `#define CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_USECOND 0`
- `#define CPPHTTPPLIB_CLIENT_READ_TIMEOUT_SECOND 300`
- `#define CPPHTTPPLIB_CLIENT_READ_TIMEOUT_USECOND 0`
- `#define CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_SECOND 5`
- `#define CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_USECOND 0`
- `#define CPPHTTPPLIB_CLIENT_MAX_TIMEOUT_MSECOND 0`
- `#define CPPHTTPPLIB_IDLE_INTERVAL_SECOND 0`

- `#define CPPHTTPPLIB_IDLE_INTERVAL_USECOND 0`
- `#define CPPHTTPPLIB_REQUEST_URI_MAX_LENGTH 8192`
- `#define CPPHTTPPLIB_HEADER_MAX_LENGTH 8192`
- `#define CPPHTTPPLIB_REDIRECT_MAX_COUNT 20`
- `#define CPPHTTPPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT 1024`
- `#define CPPHTTPPLIB_PAYLOAD_MAX_LENGTH ((std::numeric_limits<size_t>::max)())`
- `#define CPPHTTPPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH 8192`
- `#define CPPHTTPPLIB_RANGE_MAX_COUNT 1024`
- `#define CPPHTTPPLIB_TCP_NODELAY false`
- `#define CPPHTTPPLIB_IPV6_V6ONLY false`
- `#define CPPHTTPPLIB_RECV_BUFSIZ size_t(16384u)`
- `#define CPPHTTPPLIB_COMPRESSION_BUFSIZ size_t(16384u)`
- `#define CPPHTTPPLIB_THREAD_POOL_COUNT`
- `#define CPPHTTPPLIB_RECV_FLAGS 0`
- `#define CPPHTTPPLIB_SEND_FLAGS 0`
- `#define CPPHTTPPLIB_LISTEN_BACKLOG 5`
- `#define INVALID_SOCKET (-1)`
- `#define USE_IF2IP`

Определения типов

- `using socket_t = int`
- `using httpplib::Headers`
- `using httpplib::Params = std::multimap<std::string, std::string>`
- `using httpplib::Match = std::smatch`
- `using httpplib::Progress = std::function<bool(uint64_t current, uint64_t total)>`
- `using httpplib::ResponseHandler = std::function<bool(const Response &response)>`
- `using httpplib::MultipartFormDataItems = std::vector<MultipartFormData>`
- `using httpplib::MultipartFormDataMap = std::multimap<std::string, MultipartFormData>`
- `using httpplib::ContentProvider`
- `using httpplib::ContentProviderWithoutLength`
- `using httpplib::ContentProviderResourceReleaser = std::function<void(bool success)>`
- `using httpplib::MultipartFormDataProviderItems = std::vector<MultipartFormDataProvider>`
- `using httpplib::ContentReceiverWithProgress`
- `using httpplib::ContentReceiver`
- `using httpplib::MultipartContentHeader`
- `using httpplib::Range = std::pair<ssize_t, ssize_t>`
- `using httpplib::Ranges = std::vector<Range>`
- `using httpplib::Logger = std::function<void(const Request &, const Response &)>`
- `using httpplib::SocketOptions = std::function<void(socket_t sock)>`

Перечисления

- `enum httpplib::SSLVerifierResponse { httpplib::NoDecisionMade , httpplib::CertificateAccepted , httpplib::CertificateRejected }`
- `enum httpplib::StatusCode { httpplib::Continue_100 = 100 , httpplib::SwitchingProtocol_101 = 101 , httpplib::Processing_102 = 102 , httpplib::EarlyHints_103 = 103 , httpplib::OK_200 = 200 , httpplib::Created_201 = 201 , httpplib::Accepted_202 = 202 , httpplib::NonAuthoritativeInformation_203 = 203 , httpplib::NoContent_204 = 204 , httpplib::ResetContent_205 = 205 , httpplib::PartialContent_206 = 206 , httpplib::MultiStatus_207 = 207 , httpplib::AlreadyReported_208 = 208 , httpplib::IMUsed_226 = 226 , httpplib::MultipleChoices_300 = 300 , httpplib::MovedPermanently_301 = 301 ,`

```

httpplib::Found_302 = 302, httpplib::SeeOther_303 = 303, httpplib::NotModified_304 = 304,
httpplib::UseProxy_305 = 305,
httpplib::unused_306 = 306, httpplib::TemporaryRedirect_307 = 307, httpplib::PermanentRedirect_308
= 308, httpplib::BadRequest_400 = 400,
httpplib::Unauthorized_401 = 401, httpplib::PaymentRequired_402 = 402, httpplib::Forbidden_403
= 403, httpplib::NotFound_404 = 404,
httpplib::MethodNotAllowed_405 = 405, httpplib::NotAcceptable_406 = 406, httpplib::ProxyAuthenticationRequired
= 407, httpplib::RequestTimeout_408 = 408,
httpplib::Conflict_409 = 409, httpplib::Gone_410 = 410, httpplib::LengthRequired_411 = 411,
httpplib::PreconditionFailed_412 = 412,
httpplib::PayloadTooLarge_413 = 413, httpplib::UriTooLong_414 = 414, httpplib::UnsupportedMediaType_415
= 415, httpplib::RangeNotSatisfiable_416 = 416,
httpplib::ExpectationFailed_417 = 417, httpplib::ImATeapot_418 = 418, httpplib::MisdirectedRequest_421
= 421, httpplib::UnprocessableContent_422 = 422,
httpplib::Locked_423 = 423, httpplib::FailedDependency_424 = 424, httpplib::TooEarly_425 = 425
, httpplib::UpgradeRequired_426 = 426 ,
httpplib::PreconditionRequired_428 = 428, httpplib::TooManyRequests_429 = 429, httpplib::RequestHeaderFieldsToo
= 431, httpplib::UnavailableForLegalReasons_451 = 451,
httpplib::InternalServerError_500 = 500, httpplib::NotImplemented_501 = 501, httpplib::BadGateway_502
= 502, httpplib::ServiceUnavailable_503 = 503 ,
httpplib::GatewayTimeout_504 = 504 , httpplib::HttpVersionNotSupported_505 = 505 ,
httpplib::VariantAlsoNegotiates_506 = 506, httpplib::InsufficientStorage_507 = 507 ,
httpplib::LoopDetected_508 = 508, httpplib::NotExtended_510 = 510, httpplib::NetworkAuthenticationRequired_511
= 511 }
• enum class httpplib::Error {
httpplib::Success = 0, httpplib::Unknown , httpplib::Connection , httpplib::BindIPAddress ,
httpplib::Read , httpplib::Write , httpplib::ExceedRedirectCount , httpplib::Canceled ,
httpplib::SSLConnection , httpplib::SSLLoadingCerts , httpplib::SSLServerVerification , httpplib::SSLServerHostnameVer
,
httpplib::UnsupportedMultipartBoundaryChars , httpplib::Compression , httpplib::ConnectionTimeout
, httpplib::ProxyConnection ,
httpplib::SSLPeerCouldBeClosed_ }
• enum class httpplib::detail::EncodingType { httpplib::detail::None = 0 , httpplib::detail::Gzip ,
httpplib::detail::Brotli , httpplib::detail::Zstd }
```

Функции

- template<class T, class... Args>
 std::enable_if<!std::is_array< T >::value, std::unique_ptr< T >>::type `httpplib::detail::make_unique`(Args &&...args)
- template<class T>
 std::enable_if< std::is_array< T >::value, std::unique_ptr< T >>::type `httpplib::detail::make_unique`(std::size_t n)
- unsigned char `httpplib::detail::case_ignore::to_lower` (int c)
- bool `httpplib::detail::case_ignore::equal` (const std::string &a, const std::string &b)
- bool `httpplib::detail::set_socket_opt_impl` (socket_t sock, int level, int optname, const void *optval, socklen_t optlen)
- bool `httpplib::detail::set_socket_opt` (socket_t sock, int level, int optname, int opt)
- bool `httpplib::detail::set_socket_opt_time` (socket_t sock, int level, int optname, time_t sec, time_t usec)
- void `httpplib::default_socket_options` (socket_t sock)
- const char * `httpplib::status_message` (int status)
- std::string `httpplib::get_bearer_token_auth` (const Request &req)
- ssize_t `httpplib::detail::write_headers` (Stream &strm, const Headers &headers)
- std::string `httpplib::to_string` (Error error)
- std::ostream & `httpplib::operator<<` (std::ostream &os, const Error &obj)

- template<typename T, typename U>
void `httpplib::detail::duration_to_sec_and_usec` (const T &duration, U callback)
- template<size_t N>
constexpr size_t `httpplib::detail::str_len` (const char(&) [N])
- bool `httpplib::detail::is_numeric` (const std::string &str)
- uint64_t `httpplib::detail::get_header_value_u64` (const Headers &headers, const std::string &key, uint64_t def, size_t id, bool &is_invalid_value)
- uint64_t `httpplib::detail::get_header_value_u64` (const Headers &headers, const std::string &key, uint64_t def, size_t id)
- std::string `httpplib::hosted_at` (const std::string &hostname)
- void `httpplib::hosted_at` (const std::string &hostname, std::vector<std::string> &addrs)
- std::string `httpplib::append_query_params` (const std::string &path, const Params ¶ms)
- std::pair<std::string, std::string> `httpplib::make_range_header` (const Ranges &ranges)
- std::pair<std::string, std::string> `httpplib::make_basic_authentication_header` (const std::string &username, const std::string &password, bool is_proxy=false)
- std::string `httpplib::detail::encode_query_param` (const std::string &value)
- std::string `httpplib::detail::decode_url` (const std::string &s, bool convert_plus_to_space)
- std::string `httpplib::detail::trim_copy` (const std::string &s)
- void `httpplib::detail::divide` (const char *data, std::size_t size, char d, std::function<void(const char *, std::size_t, const char *, std::size_t)> fn)
- void `httpplib::detail::divide` (const std::string &str, char d, std::function<void(const char *, std::size_t, const char *, std::size_t)> fn)
- void `httpplib::detail::split` (const char *b, const char *e, char d, std::function<void(const char *, const char *)> fn)
- void `httpplib::detail::split` (const char *b, const char *e, char d, size_t m, std::function<void(const char *, const char *)> fn)
- bool `httpplib::detail::process_client_socket` (socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, time_t max_timeout_msec, std::chrono::time_point<std::chrono::steady_clock> start_time, std::function<bool(Stream &)> callback)
- socket_t `httpplib::detail::create_client_socket` (const std::string &host, const std::string &ip, int port, int address_family, bool tcp_nodelay, bool ipv6_v6only, SocketOptions socket_options, time_t connection_timeout_sec, time_t connection_timeout_usec, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, const std::string &intf, Error &error)
- const char * `httpplib::detail::get_header_value` (const Headers &headers, const std::string &key, const char *def, size_t id)
- std::string `httpplib::detail::params_to_query_str` (const Params ¶ms)
- void `httpplib::detail::parse_query_text` (const char *data, std::size_t size, Params ¶ms)
- void `httpplib::detail::parse_query_text` (const std::string &s, Params ¶ms)
- bool `httpplib::detail::parse_multipart_boundary` (const std::string &content_type, std::string &boundary)
- bool `httpplib::detail::parse_range_header` (const std::string &s, Ranges &ranges)
- int `httpplib::detail::close_socket` (socket_t sock)
- ssize_t `httpplib::detail::send_socket` (socket_t sock, const void *ptr, size_t size, int flags)
- ssize_t `httpplib::detail::read_socket` (socket_t sock, void *ptr, size_t size, int flags)
- EncodingType `httpplib::detail::encoding_type` (const Request &req, const Response &res)
- bool `httpplib::detail::fields::is_token_char` (char c)
- bool `httpplib::detail::fields::is_token` (const std::string &s)
- bool `httpplib::detail::fields::is_field_name` (const std::string &s)
- bool `httpplib::detail::fields::is_vchar` (char c)
- bool `httpplib::detail::fields::is_obs_text` (char c)
- bool `httpplib::detail::fields::is_field_vchar` (char c)
- bool `httpplib::detail::fields::is_field_content` (const std::string &s)
- bool `httpplib::detail::fields::is_field_value` (const std::string &s)
- bool `httpplib::detail::is_hex` (char c, int &v)

- `bool httpplib::detail::from_hex_to_i` (`const std::string &s, size_t i, size_t cnt, int &val`)
- `std::string httpplib::detail::from_i_to_hex` (`size_t n`)
- `size_t httpplib::detail::to_utf8` (`int code, char *buff`)
- `std::string httpplib::detail::base64_encode` (`const std::string &in`)
- `bool httpplib::detail::is_valid_path` (`const std::string &path`)
- `std::string httpplib::detail::encode_url` (`const std::string &s`)
- `std::string httpplib::detail::file_extension` (`const std::string &path`)
- `bool httpplib::detail::is_space_or_tab` (`char c`)
- `std::pair<size_t, size_t> httpplib::detail::trim` (`const char *b, const char *e, size_t left, size_t right`)
- `std::string httpplib::detail::trim_double_quotes_copy` (`const std::string &s`)
- `template<typename T>`
`ssize_t httpplib::detail::handle_EINTR` (`T fn`)
- `int httpplib::detail::poll_wrapper` (`struct pollfd *fds, nfds_t nfds, int timeout`)
- `template<bool Read>`
`ssize_t httpplib::detail::select_impl` (`socket_t sock, time_t sec, time_t usec`)
- `ssize_t httpplib::detail::select_read` (`socket_t sock, time_t sec, time_t usec`)
- `ssize_t httpplib::detail::select_write` (`socket_t sock, time_t sec, time_t usec`)
- `Error httpplib::detail::wait_until_socket_is_ready` (`socket_t sock, time_t sec, time_t usec`)
- `bool httpplib::detail::is_socket_alive` (`socket_t sock`)
- `bool httpplib::detail::keep_alive` (`const std::atomic<socket_t> &svr_sock, socket_t sock, time_t keep_alive_timeout_sec`)
- `template<typename T>`
`bool httpplib::detail::process_server_socket_core` (`const std::atomic<socket_t> &svr_sock, socket_t sock, size_t keep_alive_max_count, time_t keep_alive_timeout_sec, T callback`)
- `template<typename T>`
`bool httpplib::detail::process_server_socket` (`const std::atomic<socket_t> &svr_sock, socket_t sock, size_t keep_alive_max_count, time_t keep_alive_timeout_sec, time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec, time_t write_timeout_usec, T callback`)
- `int httpplib::detail::shutdown_socket` (`socket_t sock`)
- `std::string httpplib::detail::escape_abstract_namespace_unix_domain` (`const std::string &s`)
- `std::string httpplib::detail::unescape_abstract_namespace_unix_domain` (`const std::string &s`)
- `template<typename BindOrConnect>`
`socket_t httpplib::detail::create_socket` (`const std::string &host, const std::string &ip, int port, int address_family, int socket_flags, bool tcp_nodelay, bool ipv6_v6only, SocketOptions socket_options, BindOrConnect bind_or_connect)`
- `void httpplib::detail::set_nonblocking` (`socket_t sock, bool nonblocking`)
- `bool httpplib::detail::is_connection_error` ()
- `bool httpplib::detail::bind_ip_address` (`socket_t sock, const std::string &host`)
- `std::string httpplib::detail::if2ip` (`int address_family, const std::string &ifn`)
- `bool httpplib::detail::get_ip_and_port` (`const struct sockaddr_storage &addr, socklen_t addr_len, std::string &ip, int &port`)
- `void httpplib::detail::get_local_ip_and_port` (`socket_t sock, std::string &ip, int &port`)
- `void httpplib::detail::get_remote_ip_and_port` (`socket_t sock, std::string &ip, int &port`)
- `constexpr unsigned int httpplib::detail::str2tag_core` (`const char *s, size_t l, unsigned int h`)
- `unsigned int httpplib::detail::str2tag` (`const std::string &s`)
- `constexpr unsigned int httpplib::detail::udl::operator""_t` (`const char *s, size_t l`)
- `std::string httpplib::detail::find_content_type` (`const std::string &path, const std::map<std::string, std::string> &user_data, const std::string &default_content_type`)
- `bool httpplib::detail::can_compress_content_type` (`const std::string &content_type`)
- `bool httpplib::detail::has_header` (`const Headers &headers, const std::string &key`)
- `template<typename T>`
`bool httpplib::detail::parse_header` (`const char *beg, const char *end, T fn`)
- `bool httpplib::detail::read_headers` (`Stream &strm, Headers &headers`)
- `bool httpplib::detail::read_content_with_length` (`Stream &strm, uint64_t len, Progress progress, ContentReceiverWithProgress out`)

- void `httpplib::detail::skip_content_with_length` (`Stream &strm, uint64_t len`)
- bool `httpplib::detail::read_content_without_length` (`Stream &strm, ContentReceiverWithProgress out`)
- template<typename T>
 bool `httpplib::detail::read_content_chunked` (`Stream &strm, T &x, ContentReceiverWithProgress out`)
- bool `httpplib::detail::is_chunked_transfer_encoding` (`const Headers &headers`)
- template<typename T, typename U>
 bool `httpplib::detail::prepare_content_receiver` (`T &x, int &status, ContentReceiverWithProgress receiver, bool decompress, U callback`)
- template<typename T>
 bool `httpplib::detail::read_content` (`Stream &strm, T &x, size_t payload_max_length, int &status, Progress progress, ContentReceiverWithProgress receiver, bool decompress`)
- `ssize_t httpplib::detail::write_request_line` (`Stream &strm, const std::string &method, const std::string &path`)
- `ssize_t httpplib::detail::write_response_line` (`Stream &strm, int status`)
- bool `httpplib::detail::write_data` (`Stream &strm, const char *d, size_t l`)
- template<typename T>
 bool `httpplib::detail::write_content` (`Stream &strm, const ContentProvider &content_provider, size_t offset, size_t length, T is_shutting_down, Error &error`)
- template<typename T>
 bool `httpplib::detail::write_content` (`Stream &strm, const ContentProvider &content_provider, size_t offset, size_t length, const T &is_shutting_down`)
- template<typename T>
 bool `httpplib::detail::write_content_without_length` (`Stream &strm, const ContentProvider &content_provider, const T &is_shutting_down`)
- template<typename T, typename U>
 bool `httpplib::detail::write_content_chunked` (`Stream &strm, const ContentProvider &content_provider, const T &is_shutting_down, U &compressor, Error &error`)
- template<typename T, typename U>
 bool `httpplib::detail::write_content_chunked` (`Stream &strm, const ContentProvider &content_provider, const T &is_shutting_down, U &compressor`)
- template<typename T>
 bool `httpplib::detail::redirect` (`T &cli, Request &req, Response &res, const std::string &path, const std::string &location, Error &error`)
- void `httpplib::detail::parse_disposition_params` (`const std::string &s, Params ¶ms`)
- std::string `httpplib::detail::random_string` (`size_t length`)
- std::string `httpplib::detail::make_multipart_data_boundary` ()
- bool `httpplib::detail::is_multipart_boundary_chars_valid` (`const std::string &boundary`)
- template<typename T>
 std::string `httpplib::detail::serialize_multipart_formdata_item_begin` (`const T &item, const std::string &boundary`)
- std::string `httpplib::detail::serialize_multipart_formdata_item_end` ()
- std::string `httpplib::detail::serialize_multipart_formdata_finish` (`const std::string &boundary`)
- std::string `httpplib::detail::serialize_multipart_formdata_get_content_type` (`const std::string &boundary`)
- std::string `httpplib::detail::serialize_multipart_formdata` (`const MultipartFormDataItems &items, const std::string &boundary, bool finish=true`)
- bool `httpplib::detail::range_error` (`Request &req, Response &res`)
- std::pair< `size_t, size_t` > `httpplib::detail::get_range_offset_and_length` (`Range r, size_t content_length`)
- std::string `httpplib::detail::make_content_range_header_field` (`const std::pair< size_t, size_t > &offset_and_length, size_t content_length`)
- template<typename SToken, typename CToken, typename Content>
 bool `httpplib::detail::process_multipart_ranges_data` (`const Request &req, const std::string &boundary, const std::string &content_type, size_t content_length, SToken stoken, CToken ctoken, Content content`)

- void `httpplib::detail::make_multipart_ranges_data` (const `Request` &req, `Response` &res, const std::string &boundary, const std::string &content_type, size_t content_length, std::string &data)
- size_t `httpplib::detail::get_multipart_ranges_data_length` (const `Request` &req, const std::string &boundary, const std::string &content_type, size_t content_length)
- template<typename T>
bool `httpplib::detail::write_multipart_ranges_data` (`Stream` &strm, const `Request` &req, `Response` &res, const std::string &boundary, const std::string &content_type, size_t content_length, const T &is_shutting_down)
- bool `httpplib::detail::expect_content` (const `Request` &req)
- bool `httpplib::detail::has_crlf` (const std::string &s)
- bool `httpplib::detail::parse_www_authenticate` (const `Response` &res, std::map< std::string, std::string > &auth, bool is_proxy)
- std::pair< std::string, std::string > `httpplib::make_bearer_token_authentication_header` (const std::string &token, bool is_proxy=false)
- void `httpplib::detail::calc_actual_timeout` (time_t max_timeout_msec, time_t duration_msec, time_t timeout_sec, time_t timeout_usec, time_t &actual_timeout_sec, time_t &actual_timeout_usec)

7.7.1 Макросы

7.7.1.1 CPPHTTPLIB_CLIENT_MAX_TIMEOUT_MSECOND

```
#define CPPHTTPLIB_CLIENT_MAX_TIMEOUT_MSECOND 0
```

См. определение в файле `httpplib.h` строка 70

7.7.1.2 CPPHTTPLIB_CLIENT_READ_TIMEOUT_SECOND

```
#define CPPHTTPLIB_CLIENT_READ_TIMEOUT_SECOND 300
```

См. определение в файле `httpplib.h` строка 54

7.7.1.3 CPPHTTPLIB_CLIENT_READ_TIMEOUT_USECOND

```
#define CPPHTTPLIB_CLIENT_READ_TIMEOUT_USECOND 0
```

См. определение в файле `httpplib.h` строка 58

7.7.1.4 CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_SECOND

```
#define CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_SECOND 5
```

См. определение в файле `httpplib.h` строка 62

7.7.1.5 CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_USECOND

```
#define CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_USECOND 0
```

См. определение в файле `httpplib.h` строка 66

7.7.1.6 CPPHTTPLIB_COMPRESSION_BUFSIZ

```
#define CPPHTTPLIB_COMPRESSION_BUFSIZ size_t(16384u)
```

См. определение в файле [httpplib.h](#) строка [126](#)

7.7.1.7 CPPHTTPLIB_CONNECTION_TIMEOUT_SECOND

```
#define CPPHTTPLIB_CONNECTION_TIMEOUT_SECOND 300
```

См. определение в файле [httpplib.h](#) строка [30](#)

7.7.1.8 CPPHTTPLIB_CONNECTION_TIMEOUT_USECOND

```
#define CPPHTTPLIB_CONNECTION_TIMEOUT_USECOND 0
```

См. определение в файле [httpplib.h](#) строка [34](#)

7.7.1.9 CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH

```
#define CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH 8192
```

См. определение в файле [httpplib.h](#) строка [106](#)

7.7.1.10 CPPHTTPLIB_HEADER_MAX_LENGTH

```
#define CPPHTTPLIB_HEADER_MAX_LENGTH 8192
```

См. определение в файле [httpplib.h](#) строка [90](#)

7.7.1.11 CPPHTTPLIB_IDLE_INTERVAL_SECOND

```
#define CPPHTTPLIB_IDLE_INTERVAL_SECOND 0
```

См. определение в файле [httpplib.h](#) строка [74](#)

7.7.1.12 CPPHTTPLIB_IDLE_INTERVAL_USECOND

```
#define CPPHTTPLIB_IDLE_INTERVAL_USECOND 0
```

См. определение в файле [httpplib.h](#) строка [81](#)

7.7.1.13 CPPHTTPLIB_IPV6_V6ONLY

```
#define CPPHTTPLIB_IPV6_V6ONLY false
```

См. определение в файле [httpplib.h](#) строка [118](#)

7.7.1.14 CPPHTTPLIB_KEEPALIVE_MAX_COUNT

```
#define CPPHTTPLIB_KEEPALIVE_MAX_COUNT 100
```

См. определение в файле [httpplib.h](#) строка [26](#)

7.7.1.15 CPPHTTPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND

```
#define CPPHTTPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND 10000
```

См. определение в файле [httpplib.h](#) строка [22](#)

7.7.1.16 CPPHTTPLIB_KEEPALIVE_TIMEOUT_SECOND

```
#define CPPHTTPLIB_KEEPALIVE_TIMEOUT_SECOND 5
```

См. определение в файле [httpplib.h](#) строка [18](#)

7.7.1.17 CPPHTTPLIB_LISTEN_BACKLOG

```
#define CPPHTTPLIB_LISTEN_BACKLOG 5
```

См. определение в файле [httpplib.h](#) строка [145](#)

7.7.1.18 CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT

```
#define CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT 1024
```

См. определение в файле [httpplib.h](#) строка [98](#)

7.7.1.19 CPPHTTPLIB_PAYLOAD_MAX_LENGTH

```
#define CPPHTTPLIB_PAYLOAD_MAX_LENGTH ((std::numeric_limits<size_t>::max)())
```

См. определение в файле [httpplib.h](#) строка [102](#)

7.7.1.20 CPPHTTPLIB_RANGE_MAX_COUNT

```
#define CPPHTTPLIB_RANGE_MAX_COUNT 1024
```

См. определение в файле [httpplib.h](#) строка [110](#)

7.7.1.21 CPPHTTPLIB_RECV_BUFSIZ

```
#define CPPHTTPLIB_RECV_BUFSIZ size_t(16384u)
```

См. определение в файле [httpplib.h](#) строка [122](#)

7.7.1.22 CPPHTTPLIB_RECV_FLAGS

```
#define CPPHTTPLIB_RECV_FLAGS 0
```

См. определение в файле [httpplib.h](#) строка [137](#)

7.7.1.23 CPPHTTPLIB_REDIRECT_MAX_COUNT

```
#define CPPHTTPLIB_REDIRECT_MAX_COUNT 20
```

См. определение в файле [httpplib.h](#) строка [94](#)

7.7.1.24 CPPHTTPLIB_REQUEST_URI_MAX_LENGTH

```
#define CPPHTTPLIB_REQUEST_URI_MAX_LENGTH 8192
```

См. определение в файле [httpplib.h](#) строка [86](#)

7.7.1.25 CPPHTTPLIB_SEND_FLAGS

```
#define CPPHTTPLIB_SEND_FLAGS 0
```

См. определение в файле [httpplib.h](#) строка [141](#)

7.7.1.26 CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND

```
#define CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND 5
```

См. определение в файле [httpplib.h](#) строка [38](#)

7.7.1.27 CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND

```
#define CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND 0
```

См. определение в файле [httpplib.h](#) строка [42](#)

7.7.1.28 CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND

```
#define CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND 5
```

См. определение в файле [httpplib.h](#) строка [46](#)

7.7.1.29 CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USECOND

```
#define CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USECOND 0
```

См. определение в файле [httpplib.h](#) строка [50](#)

7.7.1.30 CPPHTTPLIB_TCP_NODELAY

```
#define CPPHTTPLIB_TCP_NODELAY false
```

См. определение в файле `httpplib.h` строка 114

7.7.1.31 CPPHTTPLIB_THREAD_POOL_COUNT

```
#define CPPHTTPLIB_THREAD_POOL_COUNT
```

Макроопределение:

```
((std::max)(8u, std::thread::hardware_concurrency() > 0  
    ? std::thread::hardware_concurrency() - 1  
    : 0)) \
```

См. определение в файле `httpplib.h` строка 130

```
00130 #define CPPHTTPLIB_THREAD_POOL_COUNT  
00131 ((std::max)(8u, std::thread::hardware_concurrency() > 0  
00132 ? std::thread::hardware_concurrency() - 1  
00133 : 0))
```

7.7.1.32 CPPHTTPLIB_VERSION

```
#define CPPHTTPLIB_VERSION "0.20.0"
```

См. определение в файле `httpplib.h` строка 11

7.7.1.33 INVALID_SOCKET

```
#define INVALID_SOCKET (-1)
```

См. определение в файле `httplib.h` строка 231

7.7.1.34 USE_IF2IP

```
#define USE_IF2IP
```

См. определение в файле [httpplib.h](#) строка 3705

7.7.2 Типы

7.7.2.1 socket t

```
using socket t = int
```

См. определение в файле `httpplib.h` строка 229

7.8 *httpplib.h*

[См. документацию.](#)

```
00001 //  
00002 // httpplib.h  
00003 //  
00004 // Copyright (c) 2025 Yuji Hirose. All rights reserved.  
00005 // MIT License  
00006 //  
00007  
00008 #ifndef CPPHTTPLIB_HTTPLIB_H  
00009 #define CPPHTTPLIB_HTTPLIB_H  
00010  
00011 #define CPPHTTPLIB_VERSION "0.20.0"  
00012  
00013 /*  
00014 * Configuration  
00015 */  
00016  
00017 #ifndef CPPHTTPLIB_KEEPALIVE_TIMEOUT_SECOND  
00018 #define CPPHTTPLIB_KEEPALIVE_TIMEOUT_SECOND 5  
00019 #endif  
00020  
00021 #ifndef CPPHTTPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND  
00022 #define CPPHTTPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND 10000  
00023 #endif  
00024  
00025 #ifndef CPPHTTPLIB_KEEPALIVE_MAX_COUNT  
00026 #define CPPHTTPLIB_KEEPALIVE_MAX_COUNT 100  
00027 #endif  
00028  
00029 #ifndef CPPHTTPLIB_CONNECTION_TIMEOUT_SECOND  
00030 #define CPPHTTPLIB_CONNECTION_TIMEOUT_SECOND 300  
00031 #endif  
00032  
00033 #ifndef CPPHTTPLIB_CONNECTION_TIMEOUT_USECOND  
00034 #define CPPHTTPLIB_CONNECTION_TIMEOUT_USECOND 0  
00035 #endif  
00036  
00037 #ifndef CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND  
00038 #define CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND 5  
00039 #endif  
00040  
00041 #ifndef CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND  
00042 #define CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND 0  
00043 #endif  
00044  
00045 #ifndef CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND  
00046 #define CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND 5  
00047 #endif  
00048  
00049 #ifndef CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USECOND  
00050 #define CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USECOND 0  
00051 #endif  
00052  
00053 #ifndef CPPHTTPLIB_CLIENT_READ_TIMEOUT_SECOND  
00054 #define CPPHTTPLIB_CLIENT_READ_TIMEOUT_SECOND 300  
00055 #endif  
00056  
00057 #ifndef CPPHTTPLIB_CLIENT_READ_TIMEOUT_USECOND  
00058 #define CPPHTTPLIB_CLIENT_READ_TIMEOUT_USECOND 0  
00059 #endif  
00060  
00061 #ifndef CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_SECOND  
00062 #define CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_SECOND 5  
00063 #endif  
00064  
00065 #ifndef CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_USECOND  
00066 #define CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_USECOND 0  
00067 #endif  
00068  
00069 #ifndef CPPHTTPLIB_CLIENT_MAX_TIMEOUT_MSECOND  
00070 #define CPPHTTPLIB_CLIENT_MAX_TIMEOUT_MSECOND 0  
00071 #endif  
00072  
00073 #ifndef CPPHTTPLIB_IDLE_INTERVAL_SECOND  
00074 #define CPPHTTPLIB_IDLE_INTERVAL_SECOND 0  
00075 #endif  
00076  
00077 #ifndef CPPHTTPLIB_IDLE_INTERVAL_USECOND  
00078 #ifdef _WIN32  
00079 #define CPPHTTPLIB_IDLE_INTERVAL_USECOND 10000  
00080 #else  
00081 #define CPPHTTPLIB_IDLE_INTERVAL_USECOND 0  
00082 #endif
```

```
00083 #endif
00084
00085 #ifndef CPPHTTPLIB_REQUEST_URI_MAX_LENGTH
00086 #define CPPHTTPLIB_REQUEST_URI_MAX_LENGTH 8192
00087 #endif
00088
00089 #ifndef CPPHTTPLIB_HEADER_MAX_LENGTH
00090 #define CPPHTTPLIB_HEADER_MAX_LENGTH 8192
00091 #endif
00092
00093 #ifndef CPPHTTPLIB_REDIRECT_MAX_COUNT
00094 #define CPPHTTPLIB_REDIRECT_MAX_COUNT 20
00095 #endif
00096
00097 #ifndef CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT
00098 #define CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT 1024
00099 #endif
00100
00101 #ifndef CPPHTTPLIB_PAYLOAD_MAX_LENGTH
00102 #define CPPHTTPLIB_PAYLOAD_MAX_LENGTH ((std::numeric_limits<size_t>::max)())
00103 #endif
00104
00105 #ifndef CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH
00106 #define CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH 8192
00107 #endif
00108
00109 #ifndef CPPHTTPLIB_RANGE_MAX_COUNT
00110 #define CPPHTTPLIB_RANGE_MAX_COUNT 1024
00111 #endif
00112
00113 #ifndef CPPHTTPLIB_TCP_NODELAY
00114 #define CPPHTTPLIB_TCP_NODELAY false
00115 #endif
00116
00117 #ifndef CPPHTTPLIB_IPV6_V6ONLY
00118 #define CPPHTTPLIB_IPV6_V6ONLY false
00119 #endif
00120
00121 #ifndef CPPHTTPLIB_RECV_BUFSIZ
00122 #define CPPHTTPLIB_RECV_BUFSIZ size_t(16384u)
00123 #endif
00124
00125 #ifndef CPPHTTPLIB_COMPRESSION_BUFSIZ
00126 #define CPPHTTPLIB_COMPRESSION_BUFSIZ size_t(16384u)
00127 #endif
00128
00129 #ifndef CPPHTTPLIB_THREAD_POOL_COUNT
00130 #define CPPHTTPLIB_THREAD_POOL_COUNT
00131 ((std::max)(8u, std::thread::hardware_concurrency() > 0
00132 ? std::thread::hardware_concurrency() - 1
00133 : 0))
00134 #endif
00135
00136 #ifndef CPPHTTPLIB_RECV_FLAGS
00137 #define CPPHTTPLIB_RECV_FLAGS 0
00138 #endif
00139
00140 #ifndef CPPHTTPLIB_SEND_FLAGS
00141 #define CPPHTTPLIB_SEND_FLAGS 0
00142 #endif
00143
00144 #ifndef CPPHTTPLIB_LISTEN_BACKLOG
00145 #define CPPHTTPLIB_LISTEN_BACKLOG 5
00146 #endif
00147
00148 /*
00149 * Headers
00150 */
00151
00152 #ifdef _WIN32
00153 #ifndef _CRT_SECURE_NO_WARNINGS
00154 #define _CRT_SECURE_NO_WARNINGS
00155 #endif // _CRT_SECURE_NO_WARNINGS
00156
00157 #ifndef _CRT_NONSTDC_NO_DEPRECATED
00158 #define _CRT_NONSTDC_NO_DEPRECATED
00159 #endif // _CRT_NONSTDC_NO_DEPRECATED
00160
00161 #if defined(_MSC_VER)
00162 #if _MSC_VER < 1900
00163 #error Sorry, Visual Studio versions prior to 2015 are not supported
00164 #endif
00165
00166 #pragma comment(lib, "ws2_32.lib")
00167
00168 #ifdef _WIN64
00169 using ssize_t = __int64;
```

```
00170 #else
00171 using ssize_t = long;
00172 #endif
00173 #endif // _MSC_VER
00174
00175 #ifndef S_ISREG
00176 #define S_ISREG(m) (((m) & S_IFREG) == S_IFREG)
00177 #endif // S_ISREG
00178
00179 #ifndef S_ISDIR
00180 #define S_ISDIR(m) (((m) & S_IFDIR) == S_IFDIR)
00181 #endif // S_ISDIR
00182
00183 #ifndef NOMINMAX
00184 #define NOMINMAX
00185 #endif // NOMINMAX
00186
00187 #include <iostream>
00188 #include <winsock2.h>
00189 #include <ws2tcpip.h>
00190
00191 // afunix.h uses types declared in winsock2.h, so has to be included after it.
00192 #include <afunix.h>
00193
00194 #ifndef WSA_FLAG_NO_HANDLE_INHERIT
00195 #define WSA_FLAG_NO_HANDLE_INHERIT 0x80
00196 #endif
00197
00198 using nfds_t = unsigned long;
00199 using socket_t = SOCKET;
00200 using socklen_t = int;
00201
00202 #else // not _WIN32
00203
00204 #include <arpa/inet.h>
00205 #if !defined(_AIX) && !defined(__MVS__)
00206 #include <ifaddrs.h>
00207 #endif
00208 #ifdef __MVS__
00209 #include <strings.h>
00210 #ifndef NI_MAXHOST
00211 #define NI_MAXHOST 1025
00212 #endif
00213 #endif
00214 #include <net/if.h>
00215 #include <netdb.h>
00216 #include <netinet/in.h>
00217 #ifdef __linux
00218 #include <resolv.h>
00219 #endif
00220 #include <csignal>
00221 #include <netinet/tcp.h>
00222 #include <poll.h>
00223 #include <pthread.h>
00224 #include <sys/mman.h>
00225 #include <sys/socket.h>
00226 #include <sys/un.h>
00227 #include <unistd.h>
00228
00229 using socket_t = int;
00230 #ifndef INVALID_SOCKET
00231 #define INVALID_SOCKET (-1)
00232 #endif
00233 #endif // _WIN32
00234
00235 #include <algorithm>
00236 #include <array>
00237 #include <atomic>
00238 #include <cassert>
00239 #include <cctype>
00240 #include <climits>
00241 #include <condition_variable>
00242 #include <cstring>
00243 #include <errno.h>
00244 #include <exception>
00245 #include <fcntl.h>
00246 #include <functional>
00247 #include <iomanip>
00248 #include <iostream>
00249 #include <list>
00250 #include <map>
00251 #include <memory>
00252 #include <mutex>
00253 #include <random>
00254 #include <regex>
00255 #include <set>
00256 #include <sstream>
```

```
00257 #include <string>
00258 #include <sys/stat.h>
00259 #include <thread>
00260 #include <unordered_map>
00261 #include <unordered_set>
00262 #include <utility>
00263
00264 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
00265 #ifndef _WIN32
00266 #include <wincrypt.h>
00267
00268 // these are defined in wincrypt.h and it breaks compilation if BoringSSL is
00269 // used
00270 #undef X509_NAME
00271 #undef X509_CERT_PAIR
00272 #undef X509_EXTENSIONS
00273 #undef PKCS7_SIGNER_INFO
00274
00275 #ifdef _MSC_VER
00276 #pragma comment(lib, "crypt32.lib")
00277 #endif
00278 #elif defined(CPPHTTPLIB_USE_CERTS_FROM_MACOSX_KEYCHAIN) && defined(__APPLE__)
00279 #include <TargetConditionals.h>
00280 #if TARGET_OS OSX
00281 #include <CoreFoundation/CoreFoundation.h>
00282 #include <Security/Security.h>
00283 #endif // TARGET_OS OSX
00284 #endif // _WIN32
00285
00286 #include <openssl/err.h>
00287 #include <openssl/evp.h>
00288 #include <openssl/ssl.h>
00289 #include <openssl/x509v3.h>
00290
00291 #if defined(_WIN32) && defined(OPENSSL_USE_APPLINK)
00292 #include <openssl/applink.c>
00293 #endif
00294
00295 #include <iostream>
00296 #include <sstream>
00297
00298 #if defined(OPENSSL_IS_BORINGSSL) || defined(LIBRESSL_VERSION_NUMBER)
00299 #if OPENSSL_VERSION_NUMBER < 0x1010107f
00300 #error Please use OpenSSL or a current version of BoringSSL
00301 #endif
00302 #define SSL_get1_peer_certificate SSL_get_peer_certificate
00303 #elif OPENSSL_VERSION_NUMBER < 0x30000000L
00304 #error Sorry, OpenSSL versions prior to 3.0.0 are not supported
00305 #endif
00306
00307 #endif
00308
00309 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
00310 #include <zlib.h>
00311 #endif
00312
00313 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
00314 #include <brotli/decode.h>
00315 #include <brotli/encode.h>
00316 #endif
00317
00318 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
00319 #include <zstd.h>
00320 #endif
00321
00322 /*
00323 * Declaration
00324 */
00325 namespace httpplib {
00326
00327 namespace detail {
00328
00329 /*
00330 * Backport std::make_unique from C++14.
00331 *
00332 * NOTE: This code came up with the following stackoverflow post:
00333 * https://stackoverflow.com/questions/10149840/c-arrays-and-make-unique
00334 *
00335 */
00336
00337 template <class T, class... Args>
00338 typename std::enable_if<!std::is_array<T>::value, std::unique_ptr<T>::type>
00339 make_unique(Args &...args) {
00340     return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
00341 }
00342
00343 template <class T>
```

```

00344 typename std::enable_if<std::is_array<T>::value, std::unique_ptr<T>::type>
00345 make_unique(std::size_t n) {
00346     typedef typename std::remove_extent<T>::type RT;
00347     return std::unique_ptr<T>(new RT[n]);
00348 }
00349
00350 namespace case_ignore {
00351
00352 inline unsigned char to_lower(int c) {
00353     const static unsigned char table[256] = {
00354         0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
00355         15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
00356         30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
00357         45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
00358         60, 61, 62, 63, 64, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
00359         107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121,
00360         122, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
00361         105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
00362         120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
00363         135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
00364         150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
00365         165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
00366         180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 224, 225, 226,
00367         227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
00368         242, 243, 244, 245, 246, 215, 248, 249, 250, 251, 252, 253, 254, 223, 224,
00369         225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
00370         240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
00371         255,
00372     };
00373     return table[(unsigned char)(char)c];
00374 }
00375
00376 inline bool equal(const std::string &a, const std::string &b) {
00377     return a.size() == b.size() &&
00378         std::equal(a.begin(), a.end(), b.begin(), [](char ca, char cb) {
00379             return to_lower(ca) == to_lower(cb);
00380         });
00381 }
00382
00383 struct equal_to {
00384     bool operator()(const std::string &a, const std::string &b) const {
00385         return equal(a, b);
00386     }
00387 };
00388
00389 struct hash {
00390     size_t operator()(const std::string &key) const {
00391         return hash_core(key.data(), key.size(), 0);
00392     }
00393
00394     size_t hash_core(const char *s, size_t l, size_t h) const {
00395         return (l == 0) ? h
00396             : hash_core(s + 1, l - 1,
00397                         // Unsets the 6 high bits of h, therefore no
00398                         // overflow happens
00399                         (((std::numeric_limits<size_t>::max()) >> 6) &
00400                         h * 33) ^
00401                         static_cast<unsigned char>(to_lower(*s)));
00402     }
00403 };
00404
00405 } // namespace case_ignore
00406
00407 // This is based on
00408 // "http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4189".
00409
00410 struct scope_exit {
00411     explicit scope_exit(std::function<void(void)> &&f)
00412         : exit_function(std::move(f)), execute_on_destruction{true} {}
00413
00414     scope_exit(scope_exit &&rhs) noexcept
00415         : exit_function(std::move(rhs.exit_function)),
00416             execute_on_destruction{rhs.execute_on_destruction} {
00417         rhs.release();
00418     }
00419
00420     ~scope_exit() {
00421         if (execute_on_destruction) { this->exit_function(); }
00422     }
00423
00424     void release() { this->execute_on_destruction = false; }
00425
00426 private:
00427     scope_exit(const scope_exit &) = delete;
00428     void operator=(const scope_exit &) = delete;
00429     scope_exit &operator=(scope_exit &&) = delete;
00430

```

```
00431 std::function<void(void)> exit_function;
00432 bool execute_on_destruction;
00433 };
00434
00435 } // namespace detail
00436
00437 enum SSLVerifierResponse {
00438 // no decision has been made, use the built-in certificate verifier
00439 NoDecisionMade,
00440 // connection certificate is verified and accepted
00441 CertificateAccepted,
00442 // connection certificate was processed but is rejected
00443 CertificateRejected
00444 };
00445
00446 enum StatusCode {
00447 // Information responses
00448 Continue_100 = 100,
00449 SwitchingProtocol_101 = 101,
00450 Processing_102 = 102,
00451 EarlyHints_103 = 103,
00452
00453 // Successful responses
00454 OK_200 = 200,
00455 Created_201 = 201,
00456 Accepted_202 = 202,
00457 NonAuthoritativeInformation_203 = 203,
00458 NoContent_204 = 204,
00459 ResetContent_205 = 205,
00460 PartialContent_206 = 206,
00461 MultiStatus_207 = 207,
00462 AlreadyReported_208 = 208,
00463 IMUsed_226 = 226,
00464
00465 // Redirection messages
00466 MultipleChoices_300 = 300,
00467 MovedPermanently_301 = 301,
00468 Found_302 = 302,
00469 SeeOther_303 = 303,
00470 NotModified_304 = 304,
00471 UseProxy_305 = 305,
00472 unused_306 = 306,
00473 TemporaryRedirect_307 = 307,
00474 PermanentRedirect_308 = 308,
00475
00476 // Client error responses
00477 BadRequest_400 = 400,
00478 Unauthorized_401 = 401,
00479 PaymentRequired_402 = 402,
00480 Forbidden_403 = 403,
00481 NotFound_404 = 404,
00482 MethodNotAllowed_405 = 405,
00483 NotAcceptable_406 = 406,
00484 ProxyAuthenticationRequired_407 = 407,
00485 RequestTimeout_408 = 408,
00486 Conflict_409 = 409,
00487 Gone_410 = 410,
00488 LengthRequired_411 = 411,
00489 PreconditionFailed_412 = 412,
00490 PayloadTooLarge_413 = 413,
00491 UriTooLong_414 = 414,
00492 UnsupportedMediaType_415 = 415,
00493 RangeNotSatisfiable_416 = 416,
00494 ExpectationFailed_417 = 417,
00495 ImATeapot_418 = 418,
00496 MisdirectedRequest_421 = 421,
00497 UnprocessableContent_422 = 422,
00498 Locked_423 = 423,
00499 FailedDependency_424 = 424,
00500 TooEarly_425 = 425,
00501 UpgradeRequired_426 = 426,
00502 PreconditionRequired_428 = 428,
00503 TooManyRequests_429 = 429,
00504 RequestHeaderFieldsTooLarge_431 = 431,
00505 UnavailableForLegalReasons_451 = 451,
00506
00507 // Server error responses
00508 InternalServerError_500 = 500,
00509 NotImplemented_501 = 501,
00510 BadGateway_502 = 502,
00511 ServiceUnavailable_503 = 503,
00512 GatewayTimeout_504 = 504,
00513 HttpVersionNotSupported_505 = 505,
00514 VariantAlsoNegotiates_506 = 506,
00515 InsufficientStorage_507 = 507,
00516 LoopDetected_508 = 508,
00517 NotExtended_510 = 510,
```

```
00518 NetworkAuthenticationRequired_511 = 511,
00519 };
00520
00521 using Headers =
00522     std::unordered_multimap<std::string, std::string, detail::case_ignore::hash,
00523                             detail::case_ignore::equal_to>;
00524
00525 using Params = std::multimap<std::string, std::string>;
00526 using Match = std::smatch;
00527
00528 using Progress = std::function<bool(uint64_t current, uint64_t total)>;
00529
00530 struct Response;
00531 using ResponseHandler = std::function<bool(const Response &response)>;
00532
00533 struct MultipartFormData {
00534     std::string name;
00535     std::string content;
00536     std::string filename;
00537     std::string content_type;
00538 };
00539 using MultipartFormDataItems = std::vector<MultipartFormData>;
00540 using MultipartFormDataMap = std::multimap<std::string, MultipartFormData>;
00541
00542 class DataSink {
00543 public:
00544     DataSink() : os(&sb_), sb_(*this) {}
00545
00546     DataSink(const DataSink &) = delete;
00547     DataSink &operator=(const DataSink &) = delete;
00548     DataSink(DataSink &&) = delete;
00549     DataSink &operator=(DataSink &&) = delete;
00550
00551     std::function<bool(const char *data, size_t data_len)> write;
00552     std::function<bool()> is_writable;
00553     std::function<void()> done;
00554     std::function<void(const Headers &trailer)> done_with_trailer;
00555     std::ostream os;
00556
00557 private:
00558     class data_sink_streambuf final : public std::streambuf {
00559     public:
00560         explicit data_sink_streambuf(DataSink &sink) : sink_(sink) {}
00561
00562     protected:
00563         std::streamsize xputn(const char *s, std::streamsize n) override {
00564             sink_.write(s, static_cast<size_t>(n));
00565             return n;
00566         }
00567
00568     private:
00569         DataSink &sink_;
00570     };
00571
00572     data_sink_streambuf sb_;
00573 };
00574
00575 using ContentProvider =
00576     std::function<bool(size_t offset, size_t length, DataSink &sink)>;
00577
00578 using ContentProviderWithoutLength =
00579     std::function<bool(size_t offset, DataSink &sink)>;
00580
00581 using ContentProviderResourceReleaser = std::function<void(bool success)>;
00582
00583 struct MultipartFormDataProvider {
00584     std::string name;
00585     ContentProviderWithoutLength provider;
00586     std::string filename;
00587     std::string content_type;
00588 };
00589 using MultipartFormDataProviderItems = std::vector<MultipartFormDataProvider>;
00590
00591 using ContentReceiverWithProgress =
00592     std::function<bool(const char *data, size_t data_length, uint64_t offset,
00593                         uint64_t total_length)>;
00594
00595 using ContentReceiver =
00596     std::function<bool(const char *data, size_t data_length)>;
00597
00598 using MultipartContentHeader =
00599     std::function<bool(const MultipartFormData &file)>;
00600
00601 class ContentReader {
00602 public:
00603     using Reader = std::function<bool(ContentReceiver receiver)>;
00604     using MultipartReader = std::function<bool(MultipartContentHeader header,
```

```
00605             ContentReceiver receiver)>;
00606
00607     ContentReader(Reader reader, MultipartReader multipart_reader)
00608     : reader_(std::move(reader)),
00609       multipart_reader_(std::move(multipart_reader)) {}
00610
00611     bool operator()(MultipartContentHeader header,
00612                       ContentReceiver receiver) const {
00613         return multipart_reader_(std::move(header), std::move(receiver));
00614     }
00615
00616     bool operator()(ContentReceiver receiver) const {
00617         return reader_(std::move(receiver));
00618     }
00619
00620     Reader reader_;
00621     MultipartReader multipart_reader_;
00622 };
00623
00624 using Range = std::pair<ssize_t, ssize_t>;
00625 using Ranges = std::vector<Range>;
00626
00627 struct Request {
00628     std::string method;
00629     std::string path;
00630     Params params;
00631     Headers headers;
00632     std::string body;
00633
00634     std::string remote_addr;
00635     int remote_port = -1;
00636     std::string local_addr;
00637     int local_port = -1;
00638
00639     // for server
00640     std::string version;
00641     std::string target;
00642     MultipartFormDataMap files;
00643     Ranges ranges;
00644     Match matches;
00645     std::unordered_map<std::string, std::string> path_params;
00646     std::function<bool()> is_connection_closed = []() { return true; };
00647
00648     // for client
00649     ResponseHandler response_handler;
00650     ContentReceiverWithProgress content_receiver;
00651     Progress progress;
00652 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
00653     const SSL *ssl = nullptr;
00654 #endif
00655
00656     bool has_header(const std::string &key) const;
00657     std::string get_header_value(const std::string &key, const char *def = "",
00658                                  size_t id = 0) const;
00659     uint64_t get_header_value_u64(const std::string &key, uint64_t def = 0,
00660                                   size_t id = 0) const;
00661     size_t get_header_value_count(const std::string &key) const;
00662     void set_header(const std::string &key, const std::string &val);
00663
00664     bool has_param(const std::string &key) const;
00665     std::string get_param_value(const std::string &key, size_t id = 0) const;
00666     size_t get_param_value_count(const std::string &key) const;
00667
00668     bool is_multipart_form_data() const;
00669
00670     bool has_file(const std::string &key) const;
00671     MultipartFormData get_file_value(const std::string &key) const;
00672     std::vector<MultipartFormData> get_file_values(const std::string &key) const;
00673
00674     // private members...
00675     size_t redirect_count_ = CPPHTTPLIB_REDIRECT_MAX_COUNT;
00676     size_t content_length_ = 0;
00677     ContentProvider content_provider_;
00678     bool is_chunked_content_provider_ = false;
00679     size_t authorization_count_ = 0;
00680     std::chrono::time_point<std::chrono::steady_clock> start_time_ =
00681         (std::chrono::steady_clock::time_point::min)();
00682 };
00683
00684 struct Response {
00685     std::string version;
00686     int status = -1;
00687     std::string reason;
00688     Headers headers;
00689     std::string body;
00690     std::string location; // Redirect location
00691
```

```
00692 bool has_header(const std::string &key) const;
00693 std::string get_header_value(const std::string &key, const char *def = "", 
00694     size_t id = 0) const;
00695 uint64_t get_header_value_u64(const std::string &key, uint64_t def = 0,
00696     size_t id = 0) const;
00697 size_t get_header_value_count(const std::string &key) const;
00698 void set_header(const std::string &key, const std::string &val);
00699
00700 void set_redirect(const std::string &url, int status = StatusCode::Found_302);
00701 void set_content(const char *s, size_t n, const std::string &content_type);
00702 void set_content(const std::string &s, const std::string &content_type);
00703 void set_content(std::string &s, const std::string &content_type);
00704
00705 void set_content_provider(
00706     size_t length, const std::string &content_type, ContentProvider provider,
00707     ContentProviderResourceReleaser resource_releaser = nullptr);
00708
00709 void set_content_provider(
00710     const std::string &content_type, ContentProviderWithoutLength provider,
00711     ContentProviderResourceReleaser resource_releaser = nullptr);
00712
00713 void set_chunked_content_provider(
00714     const std::string &content_type, ContentProviderWithoutLength provider,
00715     ContentProviderResourceReleaser resource_releaser = nullptr);
00716
00717 void set_file_content(const std::string &path,
00718     const std::string &content_type);
00719 void set_file_content(const std::string &path);
00720
00721 Response() = default;
00722 Response(const Response &) = default;
00723 Response &operator=(const Response &) = default;
00724 Response(Response &&) = default;
00725 Response &operator=(Response &&) = default;
00726 ~Response() {
00727     if (content_provider_resource_releaser_) {
00728         content_provider_resource_releaser_(content_provider_success_);
00729     }
00730 }
00731
00732 // private members...
00733 size_t content_length_ = 0;
00734 ContentProvider content_provider_;
00735 ContentProviderResourceReleaser content_provider_resource_releaser_;
00736 bool is_chunked_content_provider_ = false;
00737 bool content_provider_success_ = false;
00738 std::string file_content_path_;
00739 std::string file_content_content_type_;
00740 };
00741
00742 class Stream {
00743 public:
00744     virtual ~Stream() = default;
00745
00746     virtual bool is_readable() const = 0;
00747     virtual bool wait_readable() const = 0;
00748     virtual bool wait_writable() const = 0;
00749
00750     virtual ssize_t read(char *ptr, size_t size) = 0;
00751     virtual ssize_t write(const char *ptr, size_t size) = 0;
00752     virtual void get_remote_ip_and_port(std::string &ip, int &port) const = 0;
00753     virtual void get_local_ip_and_port(std::string &ip, int &port) const = 0;
00754     virtual socket_t socket() const = 0;
00755
00756     virtual time_t duration() const = 0;
00757
00758     ssize_t write(const char *ptr);
00759     ssize_t write(const std::string &s);
00760 };
00761
00762 class TaskQueue {
00763 public:
00764     TaskQueue() = default;
00765     virtual ~TaskQueue() = default;
00766
00767     virtual bool enqueue(std::function<void()> fn) = 0;
00768     virtual void shutdown() = 0;
00769
00770     virtual void on_idle() {}
00771 };
00772
00773 class ThreadPool final : public TaskQueue {
00774 public:
00775     explicit ThreadPool(size_t n, size_t mqr = 0)
00776         : shutdown_(false), max_queued_requests_(mqr) {
00777         while (n) {
00778             threads_.emplace_back(worker(*this));
00779         }
00780     }
00781
00782     void shutdown() {
00783         shutdown_ = true;
00784         for (auto &thread : threads_) {
00785             thread.join();
00786         }
00787     }
00788
00789     void add_task(std::function<void()> task) {
00790         enqueue(task);
00791     }
00792
00793     void join() {
00794         shutdown();
00795         for (auto &thread : threads_) {
00796             thread.join();
00797         }
00798     }
00799
00800     size_t max_queued_requests() const {
00801         return max_queued_requests_;
00802     }
00803
00804     void set_max_queued_requests(size_t mqr) {
00805         max_queued_requests_ = mqr;
00806     }
00807
00808     void set_shutdown(bool shutdown) {
00809         shutdown_ = shutdown;
00810     }
00811
00812     void set_threads(std::vector<std::thread> &threads) {
00813         threads_ = threads;
00814     }
00815
00816     std::vector<std::thread> &get_threads() {
00817         return threads_;
00818     }
00819
00820     void clear() {
00821         for (auto &thread : threads_) {
00822             thread.join();
00823         }
00824         threads_.clear();
00825     }
00826
00827     void swap(ThreadPool &other) {
00828         std::swap(shutdown_, other.shutdown_);
00829         std::swap(max_queued_requests_, other.max_queued_requests_);
00830         std::swap(threads_, other.threads_);
00831     }
00832
00833     void operator=(const ThreadPool &other) {
00834         swap(other);
00835     }
00836
00837     void operator=(ThreadPool &other) {
00838         swap(other);
00839     }
00840
00841     void operator=(const std::vector<std::thread> &threads) {
00842         threads_ = threads;
00843     }
00844
00845     void operator=(std::vector<std::thread> &threads) {
00846         threads_ = threads;
00847     }
00848
00849     void operator=(std::vector<std::thread> &threads) {
00850         threads_ = threads;
00851     }
00852
00853     void operator=(const std::vector<std::thread> &threads) {
00854         threads_ = threads;
00855     }
00856
00857     void operator=(std::vector<std::thread> &threads) {
00858         threads_ = threads;
00859     }
00860
00861     void operator=(const std::vector<std::thread> &threads) {
00862         threads_ = threads;
00863     }
00864
00865     void operator=(std::vector<std::thread> &threads) {
00866         threads_ = threads;
00867     }
00868
00869     void operator=(const std::vector<std::thread> &threads) {
00870         threads_ = threads;
00871     }
00872
00873     void operator=(std::vector<std::thread> &threads) {
00874         threads_ = threads;
00875     }
00876
00877     void operator=(const std::vector<std::thread> &threads) {
00878         threads_ = threads;
00879     }
00880
00881     void operator=(std::vector<std::thread> &threads) {
00882         threads_ = threads;
00883     }
00884
00885     void operator=(const std::vector<std::thread> &threads) {
00886         threads_ = threads;
00887     }
00888
00889     void operator=(std::vector<std::thread> &threads) {
00890         threads_ = threads;
00891     }
00892
00893     void operator=(const std::vector<std::thread> &threads) {
00894         threads_ = threads;
00895     }
00896
00897     void operator=(std::vector<std::thread> &threads) {
00898         threads_ = threads;
00899     }
00900
00901     void operator=(const std::vector<std::thread> &threads) {
00902         threads_ = threads;
00903     }
00904
00905     void operator=(std::vector<std::thread> &threads) {
00906         threads_ = threads;
00907     }
00908
00909     void operator=(const std::vector<std::thread> &threads) {
00910         threads_ = threads;
00911     }
00912
00913     void operator=(std::vector<std::thread> &threads) {
00914         threads_ = threads;
00915     }
00916
00917     void operator=(const std::vector<std::thread> &threads) {
00918         threads_ = threads;
00919     }
00920
00921     void operator=(std::vector<std::thread> &threads) {
00922         threads_ = threads;
00923     }
00924
00925     void operator=(const std::vector<std::thread> &threads) {
00926         threads_ = threads;
00927     }
00928
00929     void operator=(std::vector<std::thread> &threads) {
00930         threads_ = threads;
00931     }
00932
00933     void operator=(const std::vector<std::thread> &threads) {
00934         threads_ = threads;
00935     }
00936
00937     void operator=(std::vector<std::thread> &threads) {
00938         threads_ = threads;
00939     }
00940
00941     void operator=(const std::vector<std::thread> &threads) {
00942         threads_ = threads;
00943     }
00944
00945     void operator=(std::vector<std::thread> &threads) {
00946         threads_ = threads;
00947     }
00948
00949     void operator=(const std::vector<std::thread> &threads) {
00950         threads_ = threads;
00951     }
00952
00953     void operator=(std::vector<std::thread> &threads) {
00954         threads_ = threads;
00955     }
00956
00957     void operator=(const std::vector<std::thread> &threads) {
00958         threads_ = threads;
00959     }
00960
00961     void operator=(std::vector<std::thread> &threads) {
00962         threads_ = threads;
00963     }
00964
00965     void operator=(const std::vector<std::thread> &threads) {
00966         threads_ = threads;
00967     }
00968
00969     void operator=(std::vector<std::thread> &threads) {
00970         threads_ = threads;
00971     }
00972
00973     void operator=(const std::vector<std::thread> &threads) {
00974         threads_ = threads;
00975     }
00976
00977     void operator=(std::vector<std::thread> &threads) {
00978         threads_ = threads;
00979     }
00980
00981     void operator=(const std::vector<std::thread> &threads) {
00982         threads_ = threads;
00983     }
00984
00985     void operator=(std::vector<std::thread> &threads) {
00986         threads_ = threads;
00987     }
00988
00989     void operator=(const std::vector<std::thread> &threads) {
00990         threads_ = threads;
00991     }
00992
00993     void operator=(std::vector<std::thread> &threads) {
00994         threads_ = threads;
00995     }
00996
00997     void operator=(const std::vector<std::thread> &threads) {
00998         threads_ = threads;
00999     }
00999 }
```

```
00779     n--;
00780 }
00781 }
00782
00783 ThreadPool(const ThreadPool &) = delete;
00784 ~ThreadPool() override = default;
00785
00786 bool enqueue(std::function<void()> fn) override {
00787 {
00788     std::unique_lock<std::mutex> lock(mutex_);
00789     if (max_queued_requests_ > 0 && jobs_.size() >= max_queued_requests_) {
00790         return false;
00791     }
00792     jobs_.push_back(std::move(fn));
00793 }
00794
00795     cond_.notify_one();
00796     return true;
00797 }
00798
00799 void shutdown() override {
00800     // Stop all worker threads...
00801 {
00802     std::unique_lock<std::mutex> lock(mutex_);
00803     shutdown_ = true;
00804 }
00805
00806     cond_.notify_all();
00807
00808     // Join...
00809     for (auto &t : threads_) {
00810         t.join();
00811     }
00812 }
00813
00814 private:
00815     struct worker {
00816         explicit worker(ThreadPool &pool) : pool_(pool) {}
00817
00818         void operator()() {
00819             for (;;) {
00820                 std::function<void()> fn;
00821                 {
00822                     std::unique_lock<std::mutex> lock(pool_.mutex_);
00823
00824                     pool_.cond_.wait(
00825                         lock, [&] { return !pool_.jobs_.empty() || pool_.shutdown_; });
00826
00827                     if (pool_.shutdown_ && pool_.jobs_.empty()) { break; }
00828
00829                     fn = pool_.jobs_.front();
00830                     pool_.jobs_.pop_front();
00831                 }
00832
00833                     assert(true == static_cast<bool>(fn));
00834                     fn();
00835                 }
00836
00837 #if defined(CPPHTTPLIB_OPENSSL_SUPPORT) && !defined(OPENSSL_IS_BORINGSSL) && \
00838     !defined(LIBRESSL_VERSION_NUMBER)
00839     OpenSSL_thread_stop();
00840 #endif
00841     }
00842
00843     ThreadPool &pool_;
00844 };
00845     friend struct worker;
00846
00847     std::vector<std::thread> threads_;
00848     std::list<std::function<void()> > jobs_;
00849
00850     bool shutdown_;
00851     size_t max_queued_requests_ = 0;
00852
00853     std::condition_variable cond_;
00854     std::mutex mutex_;
00855 };
00856
00857 using Logger = std::function<void(const Request &, const Response &)>;
00858
00859 using SocketOptions = std::function<void(socket_t sock)>;
00860
00861 namespace detail {
00862
00863     bool set_socket_opt_impl(socket_t sock, int level, int optname,
00864                             const void *optval, socklen_t optlen);
00865     bool set_socket_opt(socket_t sock, int level, int optname, int opt);
```

```
00866 bool set_socket_opt_time(socket_t sock, int level, int optname, time_t sec,
00867                     time_t usec);
00868
00869 } // namespace detail
00870
00871 void default_socket_options(socket_t sock);
00872
00873 const char *status_message(int status);
00874
00875 std::string get_bearer_token_auth(const Request &req);
00876
00877 namespace detail {
00878
00879 class MatcherBase {
00880 public:
00881     virtual ~MatcherBase() = default;
00882
00883     // Match request path and populate its matches and
00884     virtual bool match(Request &request) const = 0;
00885 };
00886
00887 class PathParamsMatcher final : public MatcherBase {
00888 public:
00889     PathParamsMatcher(const std::string &pattern);
00890
00891     bool match(Request &request) const override;
00892
00893 private:
00894     // Treat segment separators as the end of path parameter capture
00895     // Does not need to handle query parameters as they are parsed before path
00896     // matching
00897     static constexpr char separator = '/';
00898
00899     // Contains static path fragments to match against, excluding the '/' after
01000     // path params
01001     // Fragments are separated by path params
01002     std::vector<std::string> static_fragments_;
01003     // Stores the names of the path parameters to be used as keys in the
01004     // Request::path_params map
01005     std::vector<std::string> param_names_;
01006 };
01007
01008 class RegexMatcher final : public MatcherBase {
01009 public:
01010     RegexMatcher(const std::string &pattern) : regex_(pattern) {}
01011
01012     bool match(Request &request) const override;
01013
01014 private:
01015     std::regex regex_;
01016
01017     ssize_t write_headers(Stream &strm, const Headers &headers);
01018
01019 } // namespace detail
01020
01021 class Server {
01022 public:
01023     using Handler = std::function<void(const Request &, Response &)>;
01024
01025     using ExceptionHandler =
01026         std::function<void(const Request &, Response &, std::exception_ptr ep)>;
01027
01028     enum class HandlerResponse {
01029         Handled,
01030         Unhandled,
01031     };
01032     using HandlerWithResponse =
01033         std::function<HandlerResponse(const Request &, Response &)>;
01034
01035     using HandlerWithContentReader = std::function<void(
01036         const Request &, Response &, const ContentReader &content_reader)>;
01037
01038     using Expect100ContinueHandler =
01039         std::function<int(const Request &, Response &)>;
01040
01041     Server();
01042
01043     virtual ~Server();
01044
01045     virtual bool is_valid() const;
01046
01047     Server &Get(const std::string &pattern, Handler handler);
01048     Server &Post(const std::string &pattern, Handler handler);
01049     Server &Post(const std::string &pattern, HandlerWithContentReader handler);
01050     Server &Put(const std::string &pattern, Handler handler);
01051     Server &Put(const std::string &pattern, HandlerWithContentReader handler);
```

```

00979 Server &Patch(const std::string &pattern, Handler handler);
00980 Server &Patch(const std::string &pattern, HandlerWithContentReader handler);
00981 Server &Delete(const std::string &pattern, Handler handler);
00982 Server &Delete(const std::string &pattern, HandlerWithContentReader handler);
00983 Server &Options(const std::string &pattern, Handler handler);
00984
00985 bool set_base_dir(const std::string &dir,
00986                     const std::string &mount_point = std::string());
00987 bool set_mount_point(const std::string &mount_point, const std::string &dir,
00988                       Headers headers = Headers());
00989 bool remove_mount_point(const std::string &mount_point);
00990 Server &set_file_extension_and_mimetype_mapping(const std::string &ext,
00991                                                 const std::string &mime);
00992 Server &set_default_file_mimetype(const std::string &mime);
00993 Server &set_file_request_handler(Handler handler);
00994
00995 template <class ErrorHandlerFunc>
00996 Server &set_error_handler(ErrorHandlerFunc &&handler) {
00997     return set_error_handler_core(
00998         std::forward<ErrorHandlerFunc>(handler),
00999         std::is_convertible<ErrorHandlerFunc, HandlerWithResponse>{});
01000 }
01001
01002 Server &set_exception_handler(ExceptionHandler handler);
01003 Server &set_pre_routing_handler(HandlerWithResponse handler);
01004 Server &set_post_routing_handler(Handler handler);
01005
01006 Server &set_expect_100_continue_handler(Expect100ContinueHandler handler);
01007 Server &set_logger(Logger logger);
01008
01009 Server &set_address_family(int family);
01010 Server &set_tcp_nodelay(bool on);
01011 Server &set_ipv6_v6only(bool on);
01012 Server &set_socket_options(SocketOptions socket_options);
01013
01014 Server &set_default_headers(Headers headers);
01015 Server &
01016 set_header_writer(std::function<ssize_t(Stream &, Headers &) > const &writer);
01017
01018 Server &set_keep_alive_max_count(size_t count);
01019 Server &set_keep_alive_timeout(time_t sec);
01020
01021 Server &set_read_timeout(time_t sec, time_t usec = 0);
01022 template <class Rep, class Period>
01023 Server &set_read_timeout(const std::chrono::duration<Rep, Period> &duration);
01024
01025 Server &set_write_timeout(time_t sec, time_t usec = 0);
01026 template <class Rep, class Period>
01027 Server &set_write_timeout(const std::chrono::duration<Rep, Period> &duration);
01028
01029 Server &set_idle_interval(time_t sec, time_t usec = 0);
01030 template <class Rep, class Period>
01031 Server &set_idle_interval(const std::chrono::duration<Rep, Period> &duration);
01032
01033 Server &set_payload_max_length(size_t length);
01034
01035 bool bind_to_port(const std::string &host, int port, int socket_flags = 0);
01036 int bind_to_any_port(const std::string &host, int socket_flags = 0);
01037 bool listen_after_bind();
01038
01039 bool listen(const std::string &host, int port, int socket_flags = 0);
01040
01041 bool is_running() const;
01042 void wait_until_ready() const;
01043 void stop();
01044 void decommission();
01045
01046 std::function<TaskQueue *(void)> new_task_queue;
01047
01048 protected:
01049     bool process_request(Stream &strm, const std::string &remote_addr,
01050                           int remote_port, const std::string &local_addr,
01051                           int local_port, bool close_connection,
01052                           bool &connection_closed,
01053                           const std::function<void(Request &)> &setup_request);
01054
01055     std::atomic<socket_t> svr_sock_{INVALID_SOCKET};
01056     size_t keep_alive_max_count_ = CPPHTTPPLIB_KEEPALIVE_MAX_COUNT;
01057     time_t keep_alive_timeout_sec_ = CPPHTTPPLIB_KEEPALIVE_TIMEOUT_SECOND;
01058     time_t read_timeout_sec_ = CPPHTTPPLIB_SERVER_READ_TIMEOUT_SECOND;
01059     time_t read_timeout_usec_ = CPPHTTPPLIB_SERVER_READ_TIMEOUT_USECOND;
01060     time_t write_timeout_sec_ = CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_SECOND;
01061     time_t write_timeout_usec_ = CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_USECOND;
01062     time_t idle_interval_sec_ = CPPHTTPPLIB_IDLE_INTERVAL_SECOND;
01063     time_t idle_interval_usec_ = CPPHTTPPLIB_IDLE_INTERVAL_USECOND;
01064     size_t payload_max_length_ = CPPHTTPPLIB_PAYLOAD_MAX_LENGTH;
01065

```

```

01066 private:
01067     using Handlers =
01068         std::vector<std::pair<std::unique_ptr<detail::MatcherBase>, Handler>;
01069     using HandlersForContentReader =
01070         std::vector<std::pair<std::unique_ptr<detail::MatcherBase>, HandlerWithContentReader>;
01071
01072
01073     static std::unique_ptr<detail::MatcherBase>
01074     make_matcher(const std::string &pattern);
01075
01076     Server &set_error_handler_core(HandlerWithResponse handler, std::true_type);
01077     Server &set_error_handler_core(Handler handler, std::false_type);
01078
01079     socket_t create_server_socket(const std::string &host, int port,
01080                               int socket_flags,
01081                               SocketOptions socket_options) const;
01082     int bind_internal(const std::string &host, int port, int socket_flags);
01083     bool listen_internal();
01084
01085     bool routing(Request &req, Response &res, Stream &strm);
01086     bool handle_file_request(const Request &req, Response &res,
01087                               bool head = false);
01088     bool dispatch_request(Request &req, Response &res,
01089                           const Handlers &handlers) const;
01090     bool dispatch_request_for_content_reader(Request &req, Response &res, ContentReader content_reader,
01091                                               const HandlersForContentReader &handlers) const;
01092
01093
01094     bool parse_request_line(const char *s, Request &req) const;
01095     void apply_ranges(const Request &req, Response &res,
01096                         std::string &content_type, std::string &boundary) const;
01097     bool write_response(Stream &strm, bool close_connection, Request &req,
01098                           Response &res);
01099     bool write_response_with_content(Stream &strm, bool close_connection,
01100                                         const Request &req, Response &res);
01101     bool write_response_core(Stream &strm, bool close_connection,
01102                               const Request &req, Response &res,
01103                               bool need_apply_ranges);
01104     bool write_content_with_provider(Stream &strm, const Request &req,
01105                                       Response &res, const std::string &boundary,
01106                                       const std::string &content_type);
01107     bool read_content(Stream &strm, Request &req, Response &res);
01108     bool
01109     read_content_with_content_receiver(Stream &strm, Request &req, Response &res,
01110                                         ContentReceiver receiver,
01111                                         MultipartContentHeader multipart_header,
01112                                         ContentReceiver multipart_receiver);
01113     bool read_content_core(Stream &strm, Request &req, Response &res,
01114                             ContentReceiver receiver,
01115                             MultipartContentHeader multipart_header,
01116                             ContentReceiver multipart_receiver) const;
01117
01118     virtual bool process_and_close_socket(socket_t sock);
01119
01120     std::atomic<bool> is_running{false};
01121     std::atomic<bool> is_decommissioned{false};
01122
01123     struct MountPointEntry {
01124         std::string mount_point;
01125         std::string base_dir;
01126         Headers headers;
01127     };
01128     std::vector<MountPointEntry> base_dirs_;
01129     std::map<std::string, std::string> file_extension_and_mimetype_map_;
01130     std::string default_file_mimetype_ = "application/octet-stream";
01131     Handler file_request_handler_;
01132
01133     Handlers get_handlers_;
01134     Handlers post_handlers_;
01135     HandlersForContentReader post_handlers_for_content_reader_;
01136     Handlers put_handlers_;
01137     HandlersForContentReader put_handlers_for_content_reader_;
01138     Handlers patch_handlers_;
01139     HandlersForContentReader patch_handlers_for_content_reader_;
01140     Handlers delete_handlers_;
01141     HandlersForContentReader delete_handlers_for_content_reader_;
01142     Handlers options_handlers_;
01143
01144     HandlerWithResponse error_handler_;
01145     ExceptionHandler exception_handler_;
01146     HandlerWithResponse pre_routing_handler_;
01147     Handler post_routing_handler_;
01148     Expect100ContinueHandler expect_100_continue_handler_;
01149
01150     Logger logger_;
01151
01152     int address_family_ = AF_UNSPEC;

```

```
01153 bool tcp_nodelay_ = CPPHTTPLIB_TCP_NODELAY;
01154 bool ipv6_v6only_ = CPPHTTPLIB_IPV6_V6ONLY;
01155 SocketOptions socket_options_ = default_socket_options;
01156
01157 Headers default_headers_;
01158 std::function<ssize_t(Stream &, Headers &)> header_writer_ =
01159     detail::write_headers;
01160 };
01161
01162 enum class Error {
01163     Success = 0,
01164     Unknown,
01165     Connection,
01166     BindIPAddress,
01167     Read,
01168     Write,
01169     ExceedRedirectCount,
01170     Canceled,
01171     SSLConnection,
01172     SSLLoadingCerts,
01173     SSLServerVerification,
01174     SSLServerHostnameVerification,
01175     UnsupportedMultipartBoundaryChars,
01176     Compression,
01177     ConnectionTimeout,
01178     ProxyConnection,
01179
01180     // For internal use only
01181     SSLPeerCouldBeClosed_,
01182 };
01183
01184 std::string to_string(Error error);
01185
01186 std::ostream &operator<<(std::ostream &os, const Error &obj);
01187
01188 class Result {
01189 public:
01190     Result() = default;
01191     Result(std::unique_ptr<Response> &&res, Error err,
01192             Headers &request_headers = Headers{});
01193     : res_(std::move(res)), err_(err),
01194       request_headers_(std::move(request_headers)) {}
01195
01196     operator bool() const { return res_ != nullptr; }
01197     bool operator==(std::nullptr_t) const { return res_ == nullptr; }
01198     bool operator!=(std::nullptr_t) const { return res_ != nullptr; }
01199     const Response &value() const { return *res_; }
01200     Response &value() { return *res_; }
01201     const Response &operator*() const { return *res_; }
01202     Response &operator*() { return *res_; }
01203     const Response *operator->() const { return res_.get(); }
01204     Response *operator->() { return res_.get(); }
01205
01206     // Error
01207     Error error() const { return err_; }
01208
01209     // Request Headers
01210     bool has_request_header(const std::string &key) const;
01211     std::string get_request_header_value(const std::string &key,
01212                                         const char *def = "",  
                                         size_t id = 0) const;
01213     uint64_t get_request_header_value_u64(const std::string &key,
01214                                         uint64_t def = 0, size_t id = 0) const;
01215     size_t get_request_header_value_count(const std::string &key) const;
01216
01217     private:
01218     std::unique_ptr<Response> res_;
01219     Error err_ = Error::Unknown;
01220     Headers request_headers_;
01221 };
01222 };
01223
01224 class ClientImpl {
01225 public:
01226     explicit ClientImpl(const std::string &host);
01227
01228     explicit ClientImpl(const std::string &host, int port);
01229
01230     explicit ClientImpl(const std::string &host, int port,
01231                          const std::string &client_cert_path,
01232                          const std::string &client_key_path);
01233
01234     virtual ~ClientImpl();
01235
01236     virtual bool is_valid() const;
01237
01238     Result Get(const std::string &path);
01239     Result Get(const std::string &path, const Headers &headers);
```

```
01240 Result Get(const std::string &path, Progress progress);
01241 Result Get(const std::string &path, const Headers &headers,
01242             Progress progress);
01243 Result Get(const std::string &path, ContentReceiver content_receiver);
01244 Result Get(const std::string &path, const Headers &headers,
01245             ContentReceiver content_receiver);
01246 Result Get(const std::string &path, ContentReceiver content_receiver,
01247             Progress progress);
01248 Result Get(const std::string &path, const Headers &headers,
01249             ContentReceiver content_receiver, Progress progress);
01250 Result Get(const std::string &path, ResponseHandler response_handler,
01251             ContentReceiver content_receiver);
01252 Result Get(const std::string &path, const Headers &headers,
01253             ResponseHandler response_handler,
01254             ContentReceiver content_receiver);
01255 Result Get(const std::string &path, ResponseHandler response_handler,
01256             ContentReceiver content_receiver, Progress progress);
01257 Result Get(const std::string &path, const Headers &headers,
01258             ResponseHandler response_handler, ContentReceiver content_receiver,
01259             Progress progress);
01260
01261 Result Get(const std::string &path, const Params &params,
01262             const Headers &headers, Progress progress = nullptr);
01263 Result Get(const std::string &path, const Params &params,
01264             const Headers &headers, ContentReceiver content_receiver,
01265             Progress progress = nullptr);
01266 Result Get(const std::string &path, const Params &params,
01267             const Headers &headers, ResponseHandler response_handler,
01268             ContentReceiver content_receiver, Progress progress = nullptr);
01269
01270 Result Head(const std::string &path);
01271 Result Head(const std::string &path, const Headers &headers);
01272
01273 Result Post(const std::string &path);
01274 Result Post(const std::string &path, const Headers &headers);
01275 Result Post(const std::string &path, const char *body, size_t content_length,
01276             const std::string &content_type);
01277 Result Post(const std::string &path, const Headers &headers, const char *body,
01278             size_t content_length, const std::string &content_type);
01279 Result Post(const std::string &path, const Headers &headers, const char *body,
01280             size_t content_length, const std::string &content_type,
01281             Progress progress);
01282 Result Post(const std::string &path, const std::string &body,
01283             const std::string &content_type);
01284 Result Post(const std::string &path, const std::string &body,
01285             const std::string &content_type, Progress progress);
01286 Result Post(const std::string &path, const Headers &headers,
01287             const std::string &body, const std::string &content_type);
01288 Result Post(const std::string &path, const Headers &headers,
01289             const std::string &body, const std::string &content_type,
01290             Progress progress);
01291 Result Post(const std::string &path, size_t content_length,
01292             ContentProvider content_provider,
01293             const std::string &content_type);
01294 Result Post(const std::string &path,
01295             ContentProviderWithoutLength content_provider,
01296             const std::string &content_type);
01297 Result Post(const std::string &path, const Headers &headers,
01298             size_t content_length, ContentProvider content_provider,
01299             const std::string &content_type);
01300 Result Post(const std::string &path, const Headers &headers,
01301             ContentProviderWithoutLength content_provider,
01302             const std::string &content_type);
01303 Result Post(const std::string &path, const Params &params);
01304 Result Post(const std::string &path, const Headers &headers,
01305             const Params &params);
01306 Result Post(const std::string &path, const Headers &headers,
01307             const Params &params, Progress progress);
01308 Result Post(const std::string &path, const MultipartFormDataItems &items);
01309 Result Post(const std::string &path, const Headers &headers,
01310             const MultipartFormDataItems &items);
01311 Result Post(const std::string &path, const Headers &headers,
01312             const MultipartFormDataItems &items, const std::string &boundary);
01313 Result Post(const std::string &path, const Headers &headers,
01314             const MultipartFormDataItems &items,
01315             const MultipartFormDataProviderItems &provider_items);
01316
01317 Result Put(const std::string &path);
01318 Result Put(const std::string &path, const char *body, size_t content_length,
01319             const std::string &content_type);
01320 Result Put(const std::string &path, const Headers &headers, const char *body,
01321             size_t content_length, const std::string &content_type);
01322 Result Put(const std::string &path, const Headers &headers, const char *body,
01323             size_t content_length, const std::string &content_type,
01324             Progress progress);
01325 Result Put(const std::string &path, const std::string &body,
01326             const std::string &content_type);
```

```
01327 Result Put(const std::string &path, const std::string &body,
01328     const std::string &content_type, Progress progress);
01329 Result Put(const std::string &path, const Headers &headers,
01330     const std::string &body, const std::string &content_type);
01331 Result Put(const std::string &path, const Headers &headers,
01332     const std::string &body, const std::string &content_type,
01333     Progress progress);
01334 Result Put(const std::string &path, size_t content_length,
01335     ContentProvider content_provider, const std::string &content_type);
01336 Result Put(const std::string &path,
01337     ContentProviderWithoutLength content_provider,
01338     const std::string &content_type);
01339 Result Put(const std::string &path, const Headers &headers,
01340     size_t content_length, ContentProvider content_provider,
01341     const std::string &content_type);
01342 Result Put(const std::string &path, const Headers &headers,
01343     ContentProviderWithoutLength content_provider,
01344     const std::string &content_type);
01345 Result Put(const std::string &path, const Params &params);
01346 Result Put(const std::string &path, const Headers &headers,
01347     const Params &params);
01348 Result Put(const std::string &path, const Headers &headers,
01349     const Params &params, Progress progress);
01350 Result Put(const std::string &path, const MultipartFormDataItems &items);
01351 Result Put(const std::string &path, const Headers &headers,
01352     const MultipartFormDataItems &items);
01353 Result Put(const std::string &path, const Headers &headers,
01354     const MultipartFormDataItems &items, const std::string &boundary);
01355 Result Put(const std::string &path, const Headers &headers,
01356     const MultipartFormDataItems &items,
01357     const MultipartFormDataProviderItems &provider_items);
01358
01359 Result Patch(const std::string &path);
01360 Result Patch(const std::string &path, const char *body, size_t content_length,
01361     const std::string &content_type);
01362 Result Patch(const std::string &path, const char *body, size_t content_length,
01363     const std::string &content_type, Progress progress);
01364 Result Patch(const std::string &path, const Headers &headers,
01365     const char *body, size_t content_length,
01366     const std::string &content_type);
01367 Result Patch(const std::string &path, const Headers &headers,
01368     const char *body, size_t content_length,
01369     const std::string &content_type, Progress progress);
01370 Result Patch(const std::string &path, const std::string &body,
01371     const std::string &content_type);
01372 Result Patch(const std::string &path, const std::string &body,
01373     const std::string &content_type, Progress progress);
01374 Result Patch(const std::string &path, const Headers &headers,
01375     const std::string &body, const std::string &content_type);
01376 Result Patch(const std::string &path, const Headers &headers,
01377     const std::string &body, const std::string &content_type,
01378     Progress progress);
01379 Result Patch(const std::string &path, size_t content_length,
01380     ContentProvider content_provider,
01381     const std::string &content_type);
01382 Result Patch(const std::string &path,
01383     ContentProviderWithoutLength content_provider,
01384     const std::string &content_type);
01385 Result Patch(const std::string &path, const Headers &headers,
01386     size_t content_length, ContentProvider content_provider,
01387     const std::string &content_type);
01388 Result Patch(const std::string &path, const Headers &headers,
01389     ContentProviderWithoutLength content_provider,
01390     const std::string &content_type);
01391
01392 Result Delete(const std::string &path);
01393 Result Delete(const std::string &path, const Headers &headers);
01394 Result Delete(const std::string &path, const char *body,
01395     size_t content_length, const std::string &content_type);
01396 Result Delete(const std::string &path, const char *body,
01397     size_t content_length, const std::string &content_type,
01398     Progress progress);
01399 Result Delete(const std::string &path, const Headers &headers,
01400     const char *body, size_t content_length,
01401     const std::string &content_type);
01402 Result Delete(const std::string &path, const Headers &headers,
01403     const char *body, size_t content_length,
01404     const std::string &content_type, Progress progress);
01405 Result Delete(const std::string &path, const std::string &body,
01406     const std::string &content_type);
01407 Result Delete(const std::string &path, const std::string &body,
01408     const std::string &content_type, Progress progress);
01409 Result Delete(const std::string &path, const Headers &headers,
01410     const std::string &body, const std::string &content_type);
01411 Result Delete(const std::string &path, const Headers &headers,
01412     const std::string &body, const std::string &content_type,
01413     Progress progress);
```

```
01414
01415     Result Options(const std::string &path);
01416     Result Options(const std::string &path, const Headers &headers);
01417
01418     bool send(Request &req, Response &res, Error &error);
01419     Result send(const Request &req);
01420
01421     void stop();
01422
01423     std::string host() const;
01424     int port() const;
01425
01426     size_t is_socket_open() const;
01427     socket_t socket() const;
01428
01429     void set_hostname_addr_map(std::map<std::string, std::string> addr_map);
01430
01431     void set_default_headers(Headers headers);
01432
01433     void
01434     set_header_writer(std::function<ssize_t(Stream &, Headers &) > const &writer);
01435
01436     void set_address_family(int family);
01437     void set_tcp_nodelay(bool on);
01438     void set_ipv6_v6only(bool on);
01439     void set_socket_options(SocketOptions socket_options);
01440
01441     void set_connection_timeout(time_t sec, time_t usec = 0);
01442     template <class Rep, class Period>
01443     void
01444     set_connection_timeout(const std::chrono::duration<Rep, Period> &duration);
01445
01446     void set_read_timeout(time_t sec, time_t usec = 0);
01447     template <class Rep, class Period>
01448     void set_read_timeout(const std::chrono::duration<Rep, Period> &duration);
01449
01450     void set_write_timeout(time_t sec, time_t usec = 0);
01451     template <class Rep, class Period>
01452     void set_write_timeout(const std::chrono::duration<Rep, Period> &duration);
01453
01454     void set_max_timeout(time_t msec);
01455     template <class Rep, class Period>
01456     void set_max_timeout(const std::chrono::duration<Rep, Period> &duration);
01457
01458     void set_basic_auth(const std::string &username, const std::string &password);
01459     void set_bearer_token_auth(const std::string &token);
01460 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01461     void set_digest_auth(const std::string &username,
01462                          const std::string &password);
01463 #endif
01464
01465     void set_keep_alive(bool on);
01466     void set_follow_location(bool on);
01467
01468     void set_url_encode(bool on);
01469
01470     void set_compress(bool on);
01471
01472     void set_decompress(bool on);
01473
01474     void set_interface(const std::string &intf);
01475
01476     void set_proxy(const std::string &host, int port);
01477     void set_proxy_basic_auth(const std::string &username,
01478                               const std::string &password);
01479     void set_proxy_bearer_token_auth(const std::string &token);
01480 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01481     void set_proxy_digest_auth(const std::string &username,
01482                               const std::string &password);
01483 #endif
01484 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01485     void set_ca_cert_path(const std::string &ca_cert_file_path,
01486                           const std::string &ca_cert_dir_path = std::string());
01487     void set_ca_cert_store(X509_STORE *ca_cert_store);
01488     X509_STORE *create_ca_cert_store(const char *ca_cert, std::size_t size) const;
01489 #endif
01490
01491 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01492     void enable_server_certificate_verification(bool enabled);
01493     void enable_server_hostname_verification(bool enabled);
01495     void set_server_certificate_verifier(
01496         std::function<SSLVerifierResponse(SSL *ssl)> verifier);
01497 #endif
01498
01499     void set_logger(Logger logger);
01500
```

```

01501 protected:
01502     struct Socket {
01503         socket_t sock = INVALID_SOCKET;
01504 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
01505         SSL *ssl = nullptr;
01506 #endif
01507     };
01508     bool is_open() const { return sock != INVALID_SOCKET; }
01509 };
01510
01511     virtual bool create_and_connect_socket(Socket &socket, Error &error);
01512
01513 // All of:
01514 // shutdown_ssl
01515 // shutdown_socket
01516 // close_socket
01517 // should ONLY be called when socket_mutex is locked.
01518 // Also, shutdown_ssl and close_socket should also NOT be called concurrently
01519 // with a DIFFERENT thread sending requests using that socket.
01520     virtual void shutdown_ssl(Socket &socket, bool shutdown_gracefully);
01521     void shutdown_socket(Socket &socket) const;
01522     void close_socket(Socket &socket);
01523
01524     bool process_request(Stream &strm, Request &req, Response &res,
01525                           bool close_connection, Error &error);
01526
01527     bool write_content_with_provider(Stream &strm, const Request &req,
01528                                     Error &error) const;
01529
01530     void copy_settings(const ClientImpl &rhs);
01531
01532 // Socket endpoint information
01533     const std::string host_;
01534     const int port_;
01535     const std::string host_and_port_;
01536
01537 // Current open socket
01538     Socket socket_;
01539     mutable std::mutex socket_mutex_;
01540     std::recursive_mutex request_mutex_;
01541
01542 // These are all protected under socket_mutex
01543     size_t socket_requests_in_flight_ = 0;
01544     std::thread::id socket_requests_are_from_thread_ = std::thread::id();
01545     bool socket_should_be_closed_when_request_is_done_ = false;
01546
01547 // Hostname-IP map
01548     std::map<std::string, std::string> addr_map_;
01549
01550 // Default headers
01551     Headers default_headers_;
01552
01553 // Header writer
01554     std::function<ssize_t(Stream &, Headers &)> header_writer_ =
01555         detail::write_headers;
01556
01557 // Settings
01558     std::string client_cert_path_;
01559     std::string client_key_path_;
01560
01561     time_t connection_timeout_sec_ = CPPHTTPPLIB_CONNECTION_TIMEOUT_SECOND;
01562     time_t connection_timeout_usec_ = CPPHTTPPLIB_CONNECTION_TIMEOUT_USECOND;
01563     time_t read_timeout_sec_ = CPPHTTPPLIB_CLIENT_READ_TIMEOUT_SECOND;
01564     time_t read_timeout_usec_ = CPPHTTPPLIB_CLIENT_READ_TIMEOUT_USECOND;
01565     time_t write_timeout_sec_ = CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_SECOND;
01566     time_t write_timeout_usec_ = CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_USECOND;
01567     time_t max_timeout_msec_ = CPPHTTPPLIB_CLIENT_MAX_TIMEOUT_MSECOND;
01568
01569     std::string basic_auth_username_;
01570     std::string basic_auth_password_;
01571     std::string bearer_token_auth_token_;
01572 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
01573     std::string digest_auth_username_;
01574     std::string digest_auth_password_;
01575 #endif
01576
01577     bool keep_alive_ = false;
01578     bool follow_location_ = false;
01579
01580     bool url_encode_ = true;
01581
01582     int address_family_ = AF_UNSPEC;
01583     bool tcp_nodelay_ = CPPHTTPPLIB_TCP_NODELAY;
01584     bool ipv6_v6only_ = CPPHTTPPLIB_IPV6_V6ONLY;
01585     SocketOptions socket_options_ = nullptr;
01586
01587     bool compress_ = false;

```

```
01588     bool decompress_ = true;
01589
01590     std::string interface_;
01591
01592     std::string proxy_host_;
01593     int proxy_port_ = -1;
01594
01595     std::string proxy_basic_auth_username_;
01596     std::string proxy_basic_auth_password_;
01597     std::string proxy_bearer_token_auth_token_;
01598 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
01599     std::string proxy_digest_auth_username_;
01600     std::string proxy_digest_auth_password_;
01601 #endif
01602
01603 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
01604     std::string ca_cert_file_path_;
01605     std::string ca_cert_dir_path_;
01606
01607     X509_STORE *ca_cert_store_ = nullptr;
01608 #endif
01609
01610 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
01611     bool server_certificate_verification_ = true;
01612     bool server_hostname_verification_ = true;
01613     std::function<SSLVerifierResponse(SSL *ssl)> server_certificate_verifier_;
01614 #endif
01615
01616     Logger logger_;
01617
01618 private:
01619     bool send_(Request &req, Response &res, Error &error);
01620     Result send_(Request &&req);
01621
01622     socket_t create_client_socket(Error &error) const;
01623     bool read_response_line(Stream &strm, const Request &req,
01624                             Response &res) const;
01625     bool write_request(Stream &strm, Request &req, bool close_connection,
01626                        Error &error);
01627     bool redirect(Request &req, Response &res, Error &error);
01628     bool handle_request(Stream &strm, Request &req, Response &res,
01629                          bool close_connection, Error &error);
01630     std::unique_ptr<Response> send_with_content_provider(
01631         Request &req, const char *body, size_t content_length,
01632         ContentProvider content_provider,
01633         ContentProviderWithoutLength content_provider_without_length,
01634         const std::string &content_type, Error &error);
01635     Result send_with_content_provider(
01636         const std::string &mmethod, const std::string &path,
01637         const Headers &headers, const char *body, size_t content_length,
01638         ContentProvider content_provider,
01639         ContentProviderWithoutLength content_provider_without_length,
01640         const std::string &content_type, Progress progress);
01641     ContentProviderWithoutLength get_multipart_content_provider(
01642         const std::string &boundary, const MultipartFormDataItems &items,
01643         const MultipartFormDataProviderItems &provider_items) const;
01644
01645     std::string adjust_host_string(const std::string &host) const;
01646
01647     virtual bool
01648     process_socket(const Socket &socket,
01649                    std::chrono::time_point<std::chrono::steady_clock> start_time,
01650                    std::function<bool(Stream &strm)> callback);
01651     virtual bool is_ssl() const;
01652 };
01653
01654 class Client {
01655 public:
01656     // Universal interface
01657     explicit Client(const std::string &scheme_host_port);
01658
01659     explicit Client(const std::string &scheme_host_port,
01660                      const std::string &client_cert_path,
01661                      const std::string &client_key_path);
01662
01663     // HTTP only interface
01664     explicit Client(const std::string &host, int port);
01665
01666     explicit Client(const std::string &host, int port,
01667                      const std::string &client_cert_path,
01668                      const std::string &client_key_path);
01669
01670     Client(Client &&) = default;
01671     Client &operator=(Client &&) = default;
01672
01673     ~Client();
01674
```

```
01675 bool is_valid() const;
01676
01677 Result Get(const std::string &path);
01678 Result Get(const std::string &path, const Headers &headers);
01679 Result Get(const std::string &path, Progress progress);
01680 Result Get(const std::string &path, const Headers &headers,
01681     Progress progress);
01682 Result Get(const std::string &path, ContentReceiver content_receiver);
01683 Result Get(const std::string &path, const Headers &headers,
01684     ContentReceiver content_receiver);
01685 Result Get(const std::string &path, ContentReceiver content_receiver,
01686     Progress progress);
01687 Result Get(const std::string &path, const Headers &headers,
01688     ContentReceiver content_receiver, Progress progress);
01689 Result Get(const std::string &path, ResponseHandler response_handler,
01690     ContentReceiver content_receiver);
01691 Result Get(const std::string &path, const Headers &headers,
01692     ResponseHandler response_handler,
01693     ContentReceiver content_receiver);
01694 Result Get(const std::string &path, const Headers &headers,
01695     ResponseHandler response_handler, ContentReceiver content_receiver,
01696     Progress progress);
01697 Result Get(const std::string &path, ResponseHandler response_handler,
01698     ContentReceiver content_receiver, Progress progress);
01699
01700 Result Get(const std::string &path, const Params &params,
01701     const Headers &headers, Progress progress = nullptr);
01702 Result Get(const std::string &path, const Params &params,
01703     const Headers &headers, ContentReceiver content_receiver,
01704     Progress progress = nullptr);
01705 Result Get(const std::string &path, const Params &params,
01706     const Headers &headers, ResponseHandler response_handler,
01707     ContentReceiver content_receiver, Progress progress = nullptr);
01708
01709 Result Head(const std::string &path);
01710 Result Head(const std::string &path, const Headers &headers);
01711
01712 Result Post(const std::string &path);
01713 Result Post(const std::string &path, const Headers &headers);
01714 Result Post(const std::string &path, const char *body, size_t content_length,
01715     const std::string &content_type);
01716 Result Post(const std::string &path, const Headers &headers, const char *body,
01717     size_t content_length, const std::string &content_type);
01718 Result Post(const std::string &path, const Headers &headers, const char *body,
01719     size_t content_length, const std::string &content_type,
01720     Progress progress);
01721 Result Post(const std::string &path, const std::string &body,
01722     const std::string &content_type);
01723 Result Post(const std::string &path, const std::string &body,
01724     const std::string &content_type, Progress progress);
01725 Result Post(const std::string &path, const Headers &headers,
01726     const std::string &body, const std::string &content_type);
01727 Result Post(const std::string &path, const Headers &headers,
01728     const std::string &body, const std::string &content_type,
01729     Progress progress);
01730 Result Post(const std::string &path, size_t content_length,
01731     ContentProvider content_provider,
01732     const std::string &content_type);
01733 Result Post(const std::string &path,
01734     ContentProviderWithoutLength content_provider,
01735     const std::string &content_type);
01736 Result Post(const std::string &path, const Headers &headers,
01737     size_t content_length, ContentProvider content_provider,
01738     const std::string &content_type);
01739 Result Post(const std::string &path, const Headers &headers,
01740     ContentProviderWithoutLength content_provider,
01741     const std::string &content_type);
01742 Result Post(const std::string &path, const Params &params);
01743 Result Post(const std::string &path, const Headers &headers,
01744     const Params &params);
01745 Result Post(const std::string &path, const Headers &headers,
01746     const Params &params, Progress progress);
01747 Result Post(const std::string &path, const MultipartFormDataItems &items);
01748 Result Post(const std::string &path, const Headers &headers,
01749     const MultipartFormDataItems &items);
01750 Result Post(const std::string &path, const Headers &headers,
01751     const MultipartFormDataItems &items, const std::string &boundary);
01752 Result Post(const std::string &path, const Headers &headers,
01753     const MultipartFormDataItems &items,
01754     const MultipartFormDataProviderItems &provider_items);
01755
01756 Result Put(const std::string &path);
01757 Result Put(const std::string &path, const char *body, size_t content_length,
01758     const std::string &content_type);
01759 Result Put(const std::string &path, const Headers &headers, const char *body,
01760     size_t content_length, const std::string &content_type);
01761 Result Put(const std::string &path, const Headers &headers, const char *body,
```

```
01762     size_t content_length, const std::string &content_type,
01763     Progress progress);
01764 Result Put(const std::string &path, const std::string &body,
01765             const std::string &content_type);
01766 Result Put(const std::string &path, const std::string &body,
01767             const std::string &content_type, Progress progress);
01768 Result Put(const std::string &path, const Headers &headers,
01769             const std::string &body, const std::string &content_type);
01770 Result Put(const std::string &path, const Headers &headers,
01771             const std::string &body, const std::string &content_type,
01772             Progress progress);
01773 Result Put(const std::string &path, size_t content_length,
01774             ContentProvider content_provider, const std::string &content_type);
01775 Result Put(const std::string &path,
01776             ContentProviderWithoutLength content_provider,
01777             const std::string &content_type);
01778 Result Put(const std::string &path, const Headers &headers,
01779             size_t content_length, ContentProvider content_provider,
01780             const std::string &content_type);
01781 Result Put(const std::string &path, const Headers &headers,
01782             ContentProviderWithoutLength content_provider,
01783             const std::string &content_type);
01784 Result Put(const std::string &path, const Params &params);
01785 Result Put(const std::string &path, const Headers &headers,
01786             const Params &params);
01787 Result Put(const std::string &path, const Headers &headers,
01788             const Params &params, Progress progress);
01789 Result Put(const std::string &path, const MultipartFormDataItems &items);
01790 Result Put(const std::string &path, const Headers &headers,
01791             const MultipartFormDataItems &items);
01792 Result Put(const std::string &path, const Headers &headers,
01793             const MultipartFormDataItems &items, const std::string &boundary);
01794 Result Put(const std::string &path, const Headers &headers,
01795             const MultipartFormDataItems &items,
01796             const MultipartFormDataProviderItems &provider_items);
01797
01798 Result Patch(const std::string &path);
01799 Result Patch(const std::string &path, const char *body, size_t content_length,
01800             const std::string &content_type);
01801 Result Patch(const std::string &path, const char *body, size_t content_length,
01802             const std::string &content_type, Progress progress);
01803 Result Patch(const std::string &path, const Headers &headers,
01804             const char *body, size_t content_length,
01805             const std::string &content_type);
01806 Result Patch(const std::string &path, const Headers &headers,
01807             const char *body, size_t content_length,
01808             const std::string &content_type, Progress progress);
01809 Result Patch(const std::string &path, const std::string &body,
01810             const std::string &content_type);
01811 Result Patch(const std::string &path, const std::string &body,
01812             const std::string &content_type, Progress progress);
01813 Result Patch(const std::string &path, const Headers &headers,
01814             const std::string &body, const std::string &content_type);
01815 Result Patch(const std::string &path, const Headers &headers,
01816             const std::string &body, const std::string &content_type,
01817             Progress progress);
01818 Result Patch(const std::string &path, size_t content_length,
01819             ContentProvider content_provider,
01820             const std::string &content_type);
01821 Result Patch(const std::string &path,
01822             ContentProviderWithoutLength content_provider,
01823             const std::string &content_type);
01824 Result Patch(const std::string &path, const Headers &headers,
01825             size_t content_length, ContentProvider content_provider,
01826             const std::string &content_type);
01827 Result Patch(const std::string &path, const Headers &headers,
01828             ContentProviderWithoutLength content_provider,
01829             const std::string &content_type);
01830
01831 Result Delete(const std::string &path);
01832 Result Delete(const std::string &path, const Headers &headers);
01833 Result Delete(const std::string &path, const char *body,
01834             size_t content_length, const std::string &content_type);
01835 Result Delete(const std::string &path, const char *body,
01836             size_t content_length, const std::string &content_type,
01837             Progress progress);
01838 Result Delete(const std::string &path, const Headers &headers,
01839             const char *body, size_t content_length,
01840             const std::string &content_type);
01841 Result Delete(const std::string &path, const Headers &headers,
01842             const char *body, size_t content_length,
01843             const std::string &content_type, Progress progress);
01844 Result Delete(const std::string &path, const std::string &body,
01845             const std::string &content_type);
01846 Result Delete(const std::string &path, const std::string &body,
01847             const std::string &content_type, Progress progress);
01848 Result Delete(const std::string &path, const Headers &headers,
```

```
01849     const std::string &body, const std::string &content_type);
01850 Result Delete(const std::string &path, const Headers &headers,
01851             const std::string &body, const std::string &content_type,
01852             Progress progress);
01853
01854 Result Options(const std::string &path);
01855 Result Options(const std::string &path, const Headers &headers);
01856
01857 bool send(Request &req, Response &res, Error &error);
01858 Result send(const Request &req);
01859
01860 void stop();
01861
01862 std::string host() const;
01863 int port() const;
01864
01865 size_t is_socket_open() const;
01866 socket_t socket() const;
01867
01868 void set_hostname_addr_map(std::map<std::string, std::string> addr_map);
01869
01870 void set_default_headers(Headers headers);
01871
01872 void
01873 set_header_writer(std::function<ssize_t(Stream &, Headers &) const &writer);
01874
01875 void set_address_family(int family);
01876 void set_tcp_nodelay(bool on);
01877 void set_socket_options(SocketOptions socket_options);
01878
01879 void set_connection_timeout(time_t sec, time_t usec = 0);
01880 template <class Rep, class Period>
01881 void
01882 set_connection_timeout(const std::chrono::duration<Rep, Period> &duration);
01883
01884 void set_read_timeout(time_t sec, time_t usec = 0);
01885 template <class Rep, class Period>
01886 void set_read_timeout(const std::chrono::duration<Rep, Period> &duration);
01887
01888 void set_write_timeout(time_t sec, time_t usec = 0);
01889 template <class Rep, class Period>
01890 void set_write_timeout(const std::chrono::duration<Rep, Period> &duration);
01891
01892 void set_max_timeout(time_t msec);
01893 template <class Rep, class Period>
01894 void set_max_timeout(const std::chrono::duration<Rep, Period> &duration);
01895
01896 void set_basic_auth(const std::string &username, const std::string &password);
01897 void set_bearer_token_auth(const std::string &token);
01898 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01899 void set_digest_auth(const std::string &username,
01900                      const std::string &password);
01901#endif
01902
01903 void set_keep_alive(bool on);
01904 void set_follow_location(bool on);
01905
01906 void set_url_encode(bool on);
01907
01908 void set_compress(bool on);
01909
01910 void set_decompress(bool on);
01911
01912 void set_interface(const std::string &ifntf);
01913
01914 void set_proxy(const std::string &host, int port);
01915 void set_proxy_basic_auth(const std::string &username,
01916                           const std::string &password);
01917 void set_proxy_bearer_token_auth(const std::string &token);
01918 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01919 void set_proxy_digest_auth(const std::string &username,
01920                           const std::string &password);
01921#endif
01922
01923 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01924 void enable_server_certificate_verification(bool enabled);
01925 void enable_server_hostname_verification(bool enabled);
01926 void set_server_certificate_verifier(
01927     std::function<SSLVerifierResponse(SSL *ssl)> verifier);
01928#endif
01929
01930 void set_logger(Logger logger);
01931
01932 // SSL
01933 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01934 void set_ca_cert_path(const std::string &ca_cert_file_path,
01935                       const std::string &ca_cert_dir_path = std::string());
01936
```

```
01936
01937 void set_ca_cert_store(X509_STORE *ca_cert_store);
01938 void load_ca_cert_store(const char *ca_cert, std::size_t size);
01939
01940 long get_openssl_verify_result() const;
01941
01942 SSL_CTX *ssl_context() const;
01943 #endif
01944
01945 private:
01946 std::unique_ptr<ClientImpl> cli_;
01947
01948 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01949 bool is_ssl_ = false;
01950 #endif
01951 };
01952
01953 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
01954 class SSLServer : public Server {
01955 public:
01956 SSLServer(const char *cert_path, const char *private_key_path,
01957             const char *client_ca_cert_file_path = nullptr,
01958             const char *client_ca_cert_dir_path = nullptr,
01959             const char *private_key_password = nullptr);
01960
01961 SSLServer(X509 *cert, EVP_PKEY *private_key,
01962             X509_STORE *client_ca_cert_store = nullptr);
01963
01964 SSLServer(
01965     const std::function<bool(SSL_CTX &ssl_ctx)> &setup_ssl_ctx_callback);
01966
01967 ~SSLServer() override;
01968
01969 bool is_valid() const override;
01970
01971 SSL_CTX *ssl_context() const;
01972
01973 void update_certs(X509 *cert, EVP_PKEY *private_key,
01974                     X509_STORE *client_ca_cert_store = nullptr);
01975
01976 private:
01977 bool process_and_close_socket(socket_t sock) override;
01978
01979 SSL_CTX *ctx_;
01980 std::mutex ctx_mutex_;
01981 };
01982
01983 class SSLClient final : public ClientImpl {
01984 public:
01985 explicit SSLClient(const std::string &host);
01986
01987 explicit SSLClient(const std::string &host, int port);
01988
01989 explicit SSLClient(const std::string &host, int port,
01990                      const std::string &client_cert_path,
01991                      const std::string &client_key_path,
01992                      const std::string &private_key_password = std::string());
01993
01994 explicit SSLClient(const std::string &host, int port, X509 *client_cert,
01995                      EVP_PKEY *client_key,
01996                      const std::string &private_key_password = std::string());
01997
01998 ~SSLClient() override;
01999
02000 bool is_valid() const override;
02001
02002 void set_ca_cert_store(X509_STORE *ca_cert_store);
02003 void load_ca_cert_store(const char *ca_cert, std::size_t size);
02004
02005 long get_openssl_verify_result() const;
02006
02007 SSL_CTX *ssl_context() const;
02008
02009 private:
02010 bool create_and_connect_socket(Socket &socket, Error &error) override;
02011 void shutdown_ssl(Socket &socket, bool shutdown_gracefully) override;
02012 void shutdown_ssl_impl(Socket &socket, bool shutdown_gracefully);
02013
02014 bool
02015 process_socket(const Socket &socket,
02016                  std::chrono::time_point<std::chrono::steady_clock> start_time,
02017                  std::function<bool(Stream &strm)> callback) override;
02018 bool is_ssl() const override;
02019
02020 bool connect_with_proxy(
02021     Socket &sock,
02022     std::chrono::time_point<std::chrono::steady_clock> start_time,
```

```

02023     Response &res, bool &success, Error &error);
02024     bool initialize_ssl(Socket &socket, Error &error);
02025
02026     bool load_certs();
02027
02028     bool verify_host(X509 *server_cert) const;
02029     bool verify_host_with_subject_alt_name(X509 *server_cert) const;
02030     bool verify_host_with_common_name(X509 *server_cert) const;
02031     bool check_host_name(const char *pattern, size_t pattern_len) const;
02032
02033     SSL_CTX *ctx_;
02034     std::mutex ctx_mutex_;
02035     std::once_flag initialize_cert_;
02036
02037     std::vector<std::string> host_components_;
02038
02039     long verify_result_ = 0;
02040
02041     friend class ClientImpl;
02042 };
02043 #endif
02044
02045 /*
02046 * Implementation of template methods.
02047 */
02048
02049 namespace detail {
02050
02051 template <typename T, typename U>
02052 inline void duration_to_sec_and_usec(const T &duration, U callback) {
02053     auto sec = std::chrono::duration_cast<std::chrono::seconds>(duration).count();
02054     auto usec = std::chrono::duration_cast<std::chrono::microseconds>(
02055         duration - std::chrono::seconds(sec))
02056         .count();
02057     callback(static_cast<time_t>(sec), static_cast<time_t>(usec));
02058 }
02059
02060 template <size_t N> inline constexpr size_t str_len(const char (&)[N]) {
02061     return N - 1;
02062 }
02063
02064 inline bool is_numeric(const std::string &str) {
02065     return !str.empty() && std::all_of(str.begin(), str.end(), ::isdigit);
02066 }
02067
02068 inline uint64_t get_header_value_u64(const Headers &headers,
02069                                         const std::string &key, uint64_t def,
02070                                         size_t id, bool &is_invalid_value) {
02071     is_invalid_value = false;
02072     auto rng = headers.equal_range(key);
02073     auto it = rng.first;
02074     std::advance(it, static_cast<ssize_t>(id));
02075     if (it != rng.second) {
02076         if (is_numeric(it->second)) {
02077             return std::strtoull(it->second.data(), nullptr, 10);
02078         } else {
02079             is_invalid_value = true;
02080         }
02081     }
02082     return def;
02083 }
02084
02085 inline uint64_t get_header_value_u64(const Headers &headers,
02086                                         const std::string &key, uint64_t def,
02087                                         size_t id) {
02088     bool dummy = false;
02089     return get_header_value_u64(headers, key, def, id, dummy);
02090 }
02091
02092 } // namespace detail
02093
02094 inline uint64_t Request::get_header_value_u64(const std::string &key,
02095                                                 uint64_t def, size_t id) const {
02096     return detail::get_header_value_u64(headers, key, def, id);
02097 }
02098
02099 inline uint64_t Response::get_header_value_u64(const std::string &key,
02100                                                 uint64_t def, size_t id) const {
02101     return detail::get_header_value_u64(headers, key, def, id);
02102 }
02103
02104 namespace detail {
02105
02106 inline bool set_socket_opt_impl(socket_t sock, int level, int optname,
02107                                     const void *optval, socklen_t optlen) {
02108     return setsockopt(sock, level, optname,
02109 #ifdef _WIN32

```

```

02110         reinterpret_cast<const char *>(optval),
02111 #else
02112         optval,
02113 #endif
02114         optlen) == 0;
02115 }
02116
02117 inline bool set_socket_opt(socket_t sock, int level, int optname, int optval) {
02118     return set_socket_opt_impl(sock, level, optname, &optval, sizeof(optval));
02119 }
02120
02121 inline bool set_socket_opt_time(socket_t sock, int level, int optname,
02122                         time_t sec, time_t usec) {
02123 #ifdef _WIN32
02124     auto timeout = static_cast<uint32_t>(sec * 1000 + usec / 1000);
02125 #else
02126     timeval timeout;
02127     timeout.tv_sec = static_cast<long>(sec);
02128     timeout.tv_usec = static_cast<decltype(timeout.tv_usec)>(usec);
02129 #endif
02130     return set_socket_opt_impl(sock, level, optname, &timeout, sizeof(timeout));
02131 }
02132
02133 } // namespace detail
02134
02135 inline void default_socket_options(socket_t sock) {
02136     detail::set_socket_opt(sock, SOL_SOCKET,
02137 #ifdef SO_REUSEPORT
02138             SO_REUSEPORT,
02139 #else
02140             SO_REUSEADDR,
02141 #endif
02142             1);
02143 }
02144
02145 inline const char *status_message(int status) {
02146     switch (status) {
02147     case StatusCode::Continue_100: return "Continue";
02148     case StatusCode::SwitchingProtocol_101: return "Switching Protocol";
02149     case StatusCode::Processing_102: return "Processing";
02150     case StatusCode::EarlyHints_103: return "Early Hints";
02151     case StatusCode::OK_200: return "OK";
02152     case StatusCode::Created_201: return "Created";
02153     case StatusCode::Accepted_202: return "Accepted";
02154     case StatusCode::NonAuthoritativeInformation_203:
02155         return "Non-Authoritative Information";
02156     case StatusCode::NoContent_204: return "No Content";
02157     case StatusCode::ResetContent_205: return "Reset Content";
02158     case StatusCode::PartialContent_206: return "Partial Content";
02159     case StatusCode::MultiStatus_207: return "Multi-Status";
02160     case StatusCode::AlreadyReported_208: return "Already Reported";
02161     case StatusCode::IMUsed_226: return "IM Used";
02162     case StatusCode::MultipleChoices_300: return "Multiple Choices";
02163     case StatusCode::MovedPermanently_301: return "Moved Permanently";
02164     case StatusCode::Found_302: return "Found";
02165     case StatusCode::SeeOther_303: return "See Other";
02166     case StatusCode::NotModified_304: return "Not Modified";
02167     case StatusCode::UseProxy_305: return "Use Proxy";
02168     case StatusCode::unused_306: return "unused";
02169     case StatusCode::TemporaryRedirect_307: return "Temporary Redirect";
02170     case StatusCode::PermanentRedirect_308: return "Permanent Redirect";
02171     case StatusCode::BadRequest_400: return "Bad Request";
02172     case StatusCode::Unauthorized_401: return "Unauthorized";
02173     case StatusCode::PaymentRequired_402: return "Payment Required";
02174     case StatusCode::Forbidden_403: return "Forbidden";
02175     case StatusCode::NotFound_404: return "Not Found";
02176     case StatusCode::MethodNotAllowed_405: return "Method Not Allowed";
02177     case StatusCode::NotAcceptable_406: return "Not Acceptable";
02178     case StatusCode::ProxyAuthenticationRequired_407:
02179         return "Proxy Authentication Required";
02180     case StatusCode::RequestTimeout_408: return "Request Timeout";
02181     case StatusCode::Conflict_409: return "Conflict";
02182     case StatusCode::Gone_410: return "Gone";
02183     case StatusCode::LengthRequired_411: return "Length Required";
02184     case StatusCode::PreconditionFailed_412: return "Precondition Failed";
02185     case StatusCode::PayloadTooLarge_413: return "Payload Too Large";
02186     case StatusCode::UriTooLong_414: return "URI Too Long";
02187     case StatusCode::UnsupportedMediaType_415: return "Unsupported Media Type";
02188     case StatusCode::RangeNotSatisfiable_416: return "Range Not Satisfiable";
02189     case StatusCode::ExpectationFailed_417: return "Expectation Failed";
02190     case StatusCode::ImATeapot_418: return "I'm a teapot";
02191     case StatusCode::MisdirectedRequest_421: return "Misdirected Request";
02192     case StatusCode::UnprocessableContent_422: return "Unprocessable Content";
02193     case StatusCode::Locked_423: return "Locked";
02194     case StatusCode::FailedDependency_424: return "Failed Dependency";
02195     case StatusCode::TooEarly_425: return "Too Early";
02196     case StatusCode::UpgradeRequired_426: return "Upgrade Required";

```

```
02197 case StatusCode::PreconditionRequired_428: return "Precondition Required";
02198 case StatusCode::TooManyRequests_429: return "Too Many Requests";
02199 case StatusCode::RequestHeaderFieldsTooLarge_431:
02200     return "Request Header Fields Too Large!";
02201 case StatusCode::UnavailableForLegalReasons_451:
02202     return "Unavailable For Legal Reasons";
02203 case StatusCode::NotImplemented_501: return "Not Implemented";
02204 case StatusCode::BadGateway_502: return "Bad Gateway";
02205 case StatusCode::ServiceUnavailable_503: return "Service Unavailable";
02206 case StatusCode::GatewayTimeout_504: return "Gateway Timeout";
02207 case StatusCode::HttpVersionNotSupported_505:
02208     return "HTTP Version Not Supported";
02209 case StatusCode::VariantAlsoNegotiates_506: return "Variant Also Negotiates";
02210 case StatusCode::InsufficientStorage_507: return "Insufficient Storage";
02211 case StatusCode::LoopDetected_508: return "Loop Detected";
02212 case StatusCode::NotExtended_510: return "Not Extended";
02213 case StatusCode::NetworkAuthenticationRequired_511:
02214     return "Network Authentication Required";
02215
02216 default:
02217 case StatusCode::InternalServerError_500: return "Internal Server Error";
02218 }
02219 }
02220
02221 inline std::string get_bearer_token_auth(const Request &req) {
02222 if (req.has_header("Authorization")) {
02223     constexpr auto bearer_header_prefix_len = detail::str_len("Bearer ");
02224     return req.get_header_value("Authorization")
02225         .substr(bearer_header_prefix_len);
02226 }
02227 return "";
02228 }
02229
02230 template <class Rep, class Period>
02231 inline Server &
02232 Server::set_read_timeout(const std::chrono::duration<Rep, Period> &duration) {
02233     detail::duration_to_sec_and_usec(
02234         duration, [&](time_t sec, time_t usec) { set_read_timeout(sec, usec); });
02235     return *this;
02236 }
02237
02238 template <class Rep, class Period>
02239 inline Server &
02240 Server::set_write_timeout(const std::chrono::duration<Rep, Period> &duration) {
02241     detail::duration_to_sec_and_usec(
02242         duration, [&](time_t sec, time_t usec) { set_write_timeout(sec, usec); });
02243     return *this;
02244 }
02245
02246 template <class Rep, class Period>
02247 inline Server &
02248 Server::set_idle_interval(const std::chrono::duration<Rep, Period> &duration) {
02249     detail::duration_to_sec_and_usec(
02250         duration, [&](time_t sec, time_t usec) { set_idle_interval(sec, usec); });
02251     return *this;
02252 }
02253
02254 inline std::string to_string(const Error error) {
02255     switch (error) {
02256     case Error::Success: return "Success (no error)";
02257     case Error::Connection: return "Could not establish connection";
02258     case Error::BindIPAddress: return "Failed to bind IP address";
02259     case Error::Read: return "Failed to read connection";
02260     case Error::Write: return "Failed to write connection";
02261     case Error::ExceedRedirectCount: return "Maximum redirect count exceeded";
02262     case Error::Canceled: return "Connection handling canceled";
02263     case Error::SSLConnection: return "SSL connection failed";
02264     case Error::SSLLoadingCerts: return "SSL certificate loading failed";
02265     case Error::SSLServerVerification: return "SSL server verification failed";
02266     case Error::SSLServerHostnameVerification:
02267         return "SSL server hostname verification failed";
02268     case Error::UnsupportedMultipartBoundaryChars:
02269         return "Unsupported HTTP multipart boundary characters";
02270     case Error::Compression: return "Compression failed";
02271     case Error::ConnectionTimeout: return "Connection timed out";
02272     case Error::ProxyConnection: return "Proxy connection failed";
02273     case Error::Unknown: return "Unknown";
02274     default: break;
02275 }
02276
02277 return "Invalid";
02278 }
02279
02280 inline std::ostream &operator<<(std::ostream &os, const Error &obj) {
02281     os << to_string(obj);
02282     os << " (" << static_cast<std::underlying_type<Error>::type>(obj) << ')';
02283     return os;
```

```
02284 }
02285
02286 inline uint64_t Result::get_request_header_value_u64(const std::string &key,
02287             uint64_t def,
02288             size_t id) const {
02289     return detail::get_header_value_u64(request_headers_, key, def, id);
02290 }
02291
02292 template <class Rep, class Period>
02293 inline void ClientImpl::set_connection_timeout(
02294     const std::chrono::duration<Rep, Period> &duration) {
02295     detail::duration_to_sec_and_usec(duration, [&](time_t sec, time_t usec) {
02296         set_connection_timeout(sec, usec);
02297     });
02298 }
02299
02300 template <class Rep, class Period>
02301 inline void ClientImpl::set_read_timeout(
02302     const std::chrono::duration<Rep, Period> &duration) {
02303     detail::duration_to_sec_and_usec(
02304         duration, [&](time_t sec, time_t usec) { set_read_timeout(sec, usec); });
02305 }
02306
02307 template <class Rep, class Period>
02308 inline void ClientImpl::set_write_timeout(
02309     const std::chrono::duration<Rep, Period> &duration) {
02310     detail::duration_to_sec_and_usec(
02311         duration, [&](time_t sec, time_t usec) { set_write_timeout(sec, usec); });
02312 }
02313
02314 template <class Rep, class Period>
02315 inline void ClientImpl::set_max_timeout(
02316     const std::chrono::duration<Rep, Period> &duration) {
02317     auto msec =
02318         std::chrono::duration_cast<std::chrono::milliseconds>(duration).count();
02319     set_max_timeout(msec);
02320 }
02321
02322 template <class Rep, class Period>
02323 inline void Client::set_connection_timeout(
02324     const std::chrono::duration<Rep, Period> &duration) {
02325     cli_->set_connection_timeout(duration);
02326 }
02327
02328 template <class Rep, class Period>
02329 inline void
02330 Client::set_read_timeout(const std::chrono::duration<Rep, Period> &duration) {
02331     cli_->set_read_timeout(duration);
02332 }
02333
02334 template <class Rep, class Period>
02335 inline void
02336 Client::set_write_timeout(const std::chrono::duration<Rep, Period> &duration) {
02337     cli_->set_write_timeout(duration);
02338 }
02339
02340 template <class Rep, class Period>
02341 inline void
02342 Client::set_max_timeout(const std::chrono::duration<Rep, Period> &duration) {
02343     cli_->set_max_timeout(duration);
02344 }
02345
02346 /*
02347 * Forward declarations and types that will be part of the .h file if split into
02348 * .h + .cc.
02349 */
02350
02351 std::string hosted_at(const std::string &hostname);
02352
02353 void hosted_at(const std::string &hostname, std::vector<std::string> &addrs);
02354
02355 std::string append_query_params(const std::string &path, const Params &params);
02356
02357 std::pair<std::string, std::string> make_range_header(const Ranges &ranges);
02358
02359 std::pair<std::string, std::string>
02360 make_basic_authentication_header(const std::string &username,
02361                                     const std::string &password,
02362                                     bool is_proxy = false);
02363
02364 namespace detail {
02365
02366 #if defined(_WIN32)
02367 inline std::wstring u8string_to_wstring(const char *s) {
02368     std::wstring ws;
02369     auto len = static_cast<int>(strlen(s));
02370     auto wlen = ::MultiByteToWideChar(CP_UTF8, 0, s, len, nullptr, 0);
02371
02372     ws.resize(wlen);
02373     ::MultiByteToWideChar(CP_UTF8, 0, s, len, ws.data(), wlen);
02374
02375     return ws;
02376 }
02377
02378 std::string wstring_to_u8string(const std::wstring &ws) {
02379     std::string s;
02380     auto len = ws.size();
02381
02382     if (len == 0) {
02383         return s;
02384     }
02385
02386     auto wlen = ::WideCharToMultiByte(CP_UTF8, 0, ws.data(), len, nullptr, 0, 0, 0);
02387
02388     s.resize(wlen);
02389     ::WideCharToMultiByte(CP_UTF8, 0, ws.data(), len, s.data(), wlen, 0, 0);
02390
02391     return s;
02392 }
```

```
02371 if (wlen > 0) {
02372     ws.resize(wlen);
02373     wlen = ::MultiByteToWideChar(
02374         CP_UTF8, 0, s, len,
02375         const_cast<LPWSTR>(reinterpret_cast<LPCWSTR>(ws.data())), wlen);
02376     if (wlen != static_cast<int>(ws.size())) { ws.clear(); }
02377 }
02378 return ws;
02379 }
02380 #endif
02381
02382 struct FileStat {
02383     FileStat(const std::string &path);
02384     bool is_file() const;
02385     bool is_dir() const;
02386
02387 private:
02388 #if defined(_WIN32)
02389     struct _stat st_;
02390 #else
02391     struct stat st_;
02392 #endif
02393     int ret_ = -1;
02394 };
02395
02396 std::string encode_query_param(const std::string &value);
02397
02398 std::string decode_url(const std::string &s, bool convert_plus_to_space);
02399
02400 std::string trim_copy(const std::string &s);
02401
02402 void divide(
02403     const char *data, std::size_t size, char d,
02404     std::function<void(const char *, std::size_t, const char *, std::size_t)>
02405         fn);
02406
02407 void divide(
02408     const std::string &str, char d,
02409     std::function<void(const char *, std::size_t, const char *, std::size_t)>
02410         fn);
02411
02412 void split(const char *b, const char *e, char d,
02413             std::function<void(const char *, const char *)> fn);
02414
02415 void split(const char *b, const char *e, char d, std::size_t m,
02416             std::function<void(const char *, const char *)> fn);
02417
02418 bool process_client_socket(
02419     socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec,
02420     time_t write_timeout_sec, time_t write_timeout_usec,
02421     time_t max_timeout_msec,
02422     std::chrono::time_point<std::chrono::steady_clock> start_time,
02423     std::function<bool(Stream &)> callback);
02424
02425 socket_t create_client_socket(const std::string &host, const std::string &ip,
02426                                 int port, int address_family, bool tcp_nodelay,
02427                                 bool ipv6_v6only, SocketOptions socket_options,
02428                                 time_t connection_timeout_sec,
02429                                 time_t connection_timeout_usec,
02430                                 time_t read_timeout_sec, time_t read_timeout_usec,
02431                                 time_t write_timeout_sec,
02432                                 time_t write_timeout_usec,
02433                                 const std::string &intf, Error &error);
02434
02435 const char *get_header_value(const Headers &headers, const std::string &key,
02436                               const char *def, size_t id);
02437
02438 std::string params_to_query_str(const Params &params);
02439
02440 void parse_query_text(const char *data, std::size_t size, Params &params);
02441
02442 void parse_query_text(const std::string &s, Params &params);
02443
02444 bool parse_multipart_boundary(const std::string &content_type,
02445                               std::string &boundary);
02446
02447 bool parse_range_header(const std::string &s, Ranges &ranges);
02448
02449 int close_socket(socket_t sock);
02450
02451 ssize_t send_socket(socket_t sock, const void *ptr, size_t size, int flags);
02452
02453 ssize_t read_socket(socket_t sock, void *ptr, size_t size, int flags);
02454
02455 enum class EncodingType { None = 0, Gzip, Brotli, Zstd };
02456
02457 EncodingType encoding_type(const Request &req, const Response &zres);
```

```
02458
02459 class BufferStream final : public Stream {
02460 public:
02461     BufferStream() = default;
02462     ~BufferStream() override = default;
02463
02464     bool is_readable() const override;
02465     bool wait_readable() const override;
02466     bool wait_writable() const override;
02467     ssize_t read(char *ptr, size_t size) override;
02468     ssize_t write(const char *ptr, size_t size) override;
02469     void get_remote_ip_and_port(std::string &ip, int &port) const override;
02470     void get_local_ip_and_port(std::string &ip, int &port) const override;
02471     socket_t socket() const override;
02472     time_t duration() const override;
02473
02474     const std::string &get_buffer() const;
02475
02476 private:
02477     std::string buffer;
02478     size_t position = 0;
02479 };
02480
02481 class compressor {
02482 public:
02483     virtual ~compressor() = default;
02484
02485     typedef std::function<bool(const char *data, size_t data_len)> Callback;
02486     virtual bool compress(const char *data, size_t data_length, bool last,
02487                           Callback callback) = 0;
02488 };
02489
02490 class decompressor {
02491 public:
02492     virtual ~decompressor() = default;
02493
02494     virtual bool is_valid() const = 0;
02495
02496     typedef std::function<bool(const char *data, size_t data_len)> Callback;
02497     virtual bool decompress(const char *data, size_t data_length,
02498                           Callback callback) = 0;
02499 };
02500
02501 class nocompressor final : public compressor {
02502 public:
02503     ~nocompressor() override = default;
02504
02505     bool compress(const char *data, size_t data_length, bool /*last*/,
02506                   Callback callback) override;
02507 };
02508
02509 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
02510 class gzip_compressor final : public compressor {
02511 public:
02512     gzip_compressor();
02513     ~gzip_compressor() override;
02514
02515     bool compress(const char *data, size_t data_length, bool last,
02516                   Callback callback) override;
02517
02518 private:
02519     bool is_valid_ = false;
02520     z_stream strm_;
02521 };
02522
02523 class gzip_decompressor final : public decompressor {
02524 public:
02525     gzip_decompressor();
02526     ~gzip_decompressor() override;
02527
02528     bool is_valid() const override;
02529
02530     bool decompress(const char *data, size_t data_length,
02531                     Callback callback) override;
02532
02533 private:
02534     bool is_valid_ = false;
02535     z_stream strm_;
02536 };
02537 #endif
02538
02539 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
02540 class brotli_compressor final : public compressor {
02541 public:
02542     brotli_compressor();
02543     ~brotli_compressor();
02544
```

```
02545 bool compress(const char *data, size_t data_length, bool last,
02546           Callback callback) override;
02547
02548 private:
02549 BrotliEncoderState *state_ = nullptr;
02550 };
02551
02552 class brotli_decompressor final : public decompressor {
02553 public:
02554 brotli_decompressor();
02555 ~brotli_decompressor();
02556
02557 bool is_valid() const override;
02558
02559 bool decompress(const char *data, size_t data_length,
02560           Callback callback) override;
02561
02562 private:
02563 BrotliDecoderResult decoder_r;
02564 BrotliDecoderState *decoder_s = nullptr;
02565 };
02566 #endif
02567
02568 #ifndef CPPHTTPLIB_ZSTD_SUPPORT
02569 class zstd_compressor : public compressor {
02570 public:
02571 zstd_compressor();
02572 ~zstd_compressor();
02573
02574 bool compress(const char *data, size_t data_length, bool last,
02575           Callback callback) override;
02576
02577 private:
02578 ZSTD_CCtx *ctx_ = nullptr;
02579 };
02580
02581 class zstd_decompressor : public decompressor {
02582 public:
02583 zstd_decompressor();
02584 ~zstd_decompressor();
02585
02586 bool is_valid() const override;
02587
02588 bool decompress(const char *data, size_t data_length,
02589           Callback callback) override;
02590
02591 private:
02592 ZSTD_DCtx *ctx_ = nullptr;
02593 };
02594 #endif
02595
02596 // NOTE: until the read size reaches `fixed_buffer_size`, use `fixed_buffer`
02597 // to store data. The call can set memory on stack for performance.
02598 class stream_line_reader {
02599 public:
02600 stream_line_reader(Stream &strm, char *fixed_buffer,
02601           size_t fixed_buffer_size);
02602 const char *ptr() const;
02603 size_t size() const;
02604 bool end_with_crlf() const;
02605 bool getline();
02606
02607 private:
02608 void append(char c);
02609
02610 Stream &strm_;
02611 char *fixed_buffer_;
02612 const size_t fixed_buffer_size_;
02613 size_t fixed_buffer_used_size_ = 0;
02614 std::string growable_buffer_;
02615 };
02616
02617 class mmap {
02618 public:
02619 mmap(const char *path);
02620 ~mmap();
02621
02622 bool open(const char *path);
02623 void close();
02624
02625 bool is_open() const;
02626 size_t size() const;
02627 const char *data() const;
02628
02629 private:
02630 #if defined(_WIN32)
02631 HANDLE hFile_ = NULL;
```

```

02632 HANDLE hMapping_ = NULL;
02633 #else
02634 int fd_ = -1;
02635 #endif
02636 size_t size_ = 0;
02637 void*addr_ = nullptr;
02638 bool is_open_empty_file = false;
02639 };
02640
02641 // NOTE: https://www.rfc-editor.org/rfc/rfc9110#section-5
02642 namespace fields {
02643
02644 inline bool is_token_char(char c) {
02645     return std::isalnum(c) || c == '!' || c == '#' || c == '$' || c == '%' ||
02646         c == '&' || c == '^' || c == '*' || c == '+' || c == '_' ||
02647         c == '.' || c == ',' || c == '_' || c == ';' || c == ']' || c == '-';
02648 }
02649
02650 inline bool is_token(const std::string &s) {
02651     if (s.empty()) { return false; }
02652     for (auto c : s) {
02653         if (!is_token_char(c)) { return false; }
02654     }
02655     return true;
02656 }
02657
02658 inline bool is_field_name(const std::string &s) { return is_token(s); }
02659
02660 inline bool is_vchar(char c) { return c >= 33 && c <= 126; }
02661
02662 inline bool is_obs_text(char c) { return 128 <= static_cast<unsigned char>(c); }
02663
02664 inline bool is_field_vchar(char c) { return is_vchar(c) || is_obs_text(c); }
02665
02666 inline bool is_field_content(const std::string &s) {
02667     if (s.empty()) { return true; }
02668
02669     if (s.size() == 1) {
02670         return is_field_vchar(s[0]);
02671     } else if (s.size() == 2) {
02672         return is_field_vchar(s[0]) && is_field_vchar(s[1]);
02673     } else {
02674         size_t i = 0;
02675
02676         if (!is_field_vchar(s[i])) { return false; }
02677         i++;
02678
02679         while (i < s.size() - 1) {
02680             auto c = s[i++];
02681             if (c == ' ' || c == '\t' || is_field_vchar(c)) {
02682                 } else {
02683                     return false;
02684                 }
02685             }
02686
02687         return is_field_vchar(s[i]);
02688     }
02689 }
02690
02691 inline bool is_field_value(const std::string &s) { return is_field_content(s); }
02692
02693 } // namespace fields
02694
02695 } // namespace detail
02696
02697 // -----
02698 /*
02699 * Implementation that will be part of the .cc file if split into .h + .cc.
02700 */
02701
02702
02703 namespace detail {
02704
02705 inline bool is_hex(char c, int &v) {
02706     if (0x20 <= c && isdigit(c)) {
02707         v = c - '0';
02708         return true;
02709     } else if ('A' <= c && c <= 'F') {
02710         v = c - 'A' + 10;
02711         return true;
02712     } else if ('a' <= c && c <= 'f') {
02713         v = c - 'a' + 10;
02714         return true;
02715     }
02716     return false;
02717 }
02718

```

```

02719 inline bool from_hex_to_i(const std::string &s, size_t i, size_t cnt,
02720     int &val) {
02721     if (i >= s.size()) { return false; }
02722
02723     val = 0;
02724     for (; cnt; i++, cnt--) {
02725         if (!s[i]) { return false; }
02726         auto v = 0;
02727         if (is_hex(s[i], v)) {
02728             val = val * 16 + v;
02729         } else {
02730             return false;
02731         }
02732     }
02733     return true;
02734 }
02735
02736 inline std::string from_i_to_hex(size_t n) {
02737     static const auto charset = "0123456789abcdef";
02738     std::string ret;
02739     do {
02740         ret = charset[n & 15] + ret;
02741         n >>= 4;
02742     } while (n > 0);
02743     return ret;
02744 }
02745
02746 inline size_t to_utf8(int code, char *buff) {
02747     if (code < 0x080) {
02748         buff[0] = static_cast<char>(code & 0x7F);
02749         return 1;
02750     } else if (code < 0x0800) {
02751         buff[0] = static_cast<char>(0xC0 | ((code >> 6) & 0x1F));
02752         buff[1] = static_cast<char>(0x80 | (code & 0x3F));
02753         return 2;
02754     } else if (code < 0xD800) {
02755         buff[0] = static_cast<char>(0xE0 | ((code >> 12) & 0xF));
02756         buff[1] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02757         buff[2] = static_cast<char>(0x80 | (code & 0x3F));
02758         return 3;
02759     } else if (code < 0xE000) { // D800 - DFFF is invalid...
02760         return 0;
02761     } else if (code < 0x10000) {
02762         buff[0] = static_cast<char>(0xE0 | ((code >> 12) & 0xF));
02763         buff[1] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02764         buff[2] = static_cast<char>(0x80 | (code & 0x3F));
02765         return 3;
02766     } else if (code < 0x110000) {
02767         buff[0] = static_cast<char>(0xF0 | ((code >> 18) & 0x7));
02768         buff[1] = static_cast<char>(0x80 | ((code >> 12) & 0x3F));
02769         buff[2] = static_cast<char>(0x80 | ((code >> 6) & 0x3F));
02770         buff[3] = static_cast<char>(0x80 | (code & 0x3F));
02771         return 4;
02772     }
02773
02774 // NOTREACHED
02775     return 0;
02776 }
02777
02778 // NOTE: This code came up with the following stackoverflow post:
02779 // https://stackoverflow.com/questions/180947/base64-decode-snippet-in-c
02780 inline std::string base64_encode(const std::string &in) {
02781     static const auto lookup =
02782         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
02783
02784     std::string out;
02785     out.reserve(in.size());
02786
02787     auto val = 0;
02788     auto valb = -6;
02789
02790     for (auto c : in) {
02791         val = (val << 8) + static_cast<uint8_t>(c);
02792         valb += 8;
02793         while (valb >= 0) {
02794             out.push_back(lookup[(val >> valb) & 0x3F]);
02795             valb -= 6;
02796         }
02797     }
02798     if (valb > -6) { out.push_back(lookup[((val << 8) >> (valb + 8)) & 0x3F]); }
02800
02801     while (out.size() % 4) {
02802         out.push_back('=');
02803     }
02804
02805     return out;

```

```
02806 }
02807
02808 inline bool is_valid_path(const std::string &path) {
02809     size_t level = 0;
02810     size_t i = 0;
02811
02812     // Skip slash
02813     while (i < path.size() && path[i] == '/') {
02814         i++;
02815     }
02816
02817     while (i < path.size()) {
02818         // Read component
02819         auto beg = i;
02820         while (i < path.size() && path[i] != '/') {
02821             if (path[i] == '\0') {
02822                 return false;
02823             } else if (path[i] == '\\') {
02824                 return false;
02825             }
02826             i++;
02827         }
02828
02829         auto len = i - beg;
02830         assert(len > 0);
02831
02832         if (!path.compare(beg, len, "."))
02833             ;
02834         } else if (!path.compare(beg, len, "..")) {
02835             if (level == 0) { return false; }
02836             level--;
02837         } else {
02838             level++;
02839         }
02840
02841     // Skip slash
02842     while (i < path.size() && path[i] == '/') {
02843         i++;
02844     }
02845 }
02846
02847 return true;
02848 }
02849
02850 inline FileStat::FileStat(const std::string &path) {
02851 #if defined(_WIN32)
02852     auto wpath = u8string_to_wstring(path.c_str());
02853     ret_ = _wstat(wpath.c_str(), &st_);
02854 #else
02855     ret_ = stat(path.c_str(), &st_);
02856 #endif
02857 }
02858 inline bool FileStat::is_file() const {
02859     return ret_ >= 0 && S_ISREG(st_.st_mode);
02860 }
02861 inline bool FileStat::is_dir() const {
02862     return ret_ >= 0 && S_ISDIR(st_.st_mode);
02863 }
02864
02865 inline std::string encode_query_param(const std::string &value) {
02866     std::ostringstream escaped;
02867     escaped.fill('0');
02868     escaped << std::hex;
02869
02870     for (auto c : value) {
02871         if (std::isalnum(static_cast<uint8_t>(c)) || c == '-' || c == '_' ||
02872             c == '.' || c == '!' || c == '~' || c == '*' || c == '\'' || c == '(' ||
02873             c == ')') {
02874             escaped << c;
02875         } else {
02876             escaped << std::uppercase;
02877             escaped << '%' << std::setw(2)
02878                 << static_cast<int>(static_cast<unsigned char>(c));
02879             escaped << std::nouppercase;
02880         }
02881     }
02882
02883     return escaped.str();
02884 }
02885
02886 inline std::string encode_url(const std::string &s) {
02887     std::string result;
02888     result.reserve(s.size());
02889
02890     for (size_t i = 0; s[i]; i++) {
02891         switch (s[i]) {
02892             case ':': result += "%20"; break;
```

```

02893     case '+': result += "%2B"; break;
02894     case '\r': result += "%0D"; break;
02895     case '\n': result += "%0A"; break;
02896     case '\v': result += "%27"; break;
02897     case ',': result += "%2C"; break;
02898     // case ?: result += "%3A"; break; // ok? probably...
02899     case ';': result += "%3B"; break;
02900     default:
02901         auto c = static_cast<uint8_t>(s[i]);
02902         if (c >= 0x80) {
02903             result += '%';
02904             char hex[4];
02905             auto len = sprintf(hex, sizeof(hex) - 1, "%02X", c);
02906             assert(len == 2);
02907             result.append(hex, static_cast<size_t>(len));
02908         } else {
02909             result += s[i];
02910         }
02911         break;
02912     }
02913 }
02914
02915 return result;
02916 }
02917
02918 inline std::string decode_url(const std::string &s,
02919                               bool convert_plus_to_space) {
02920     std::string result;
02921
02922     for (size_t i = 0; i < s.size(); i++) {
02923         if (s[i] == '%' && i + 1 < s.size()) {
02924             if (s[i + 1] == 'u') {
02925                 auto val = 0;
02926                 if (from_hex_to_i(s, i + 2, 4, val)) {
02927                     // 4 digits Unicode codes
02928                     char buff[4];
02929                     size_t len = to_utf8(val, buff);
02930                     if (len > 0) { result.append(buff, len); }
02931                     i += 5; // 'u0000'
02932                 } else {
02933                     result += s[i];
02934                 }
02935             } else {
02936                 auto val = 0;
02937                 if (from_hex_to_i(s, i + 1, 2, val)) {
02938                     result += static_cast<char>(val);
02939                     i += 2; // '00'
02940                 } else {
02941                     result += s[i];
02942                 }
02943             }
02944         } else if (convert_plus_to_space && s[i] == '+') {
02945             result += ' ';
02946         } else {
02947             result += s[i];
02948         }
02949     }
02950 }
02951
02952 return result;
02953 }
02954
02955 inline std::string file_extension(const std::string &path) {
02956     std::smatch m;
02957     thread_local auto re = std::regex("\\.(\\w+)$");
02958     if (std::regex_search(path, m, re)) { return m[1].str(); }
02959     return std::string();
02960 }
02961
02962 inline bool is_space_or_tab(char c) { return c == ' ' || c == '\t'; }
02963
02964 inline std::pair<size_t, size_t> trim(const char *b, const char *e, size_t left,
02965                                         size_t right) {
02966     while (b + left < e && is_space_or_tab(b[left])) {
02967         left++;
02968     }
02969     while (right > 0 && is_space_or_tab(b[right - 1])) {
02970         right--;
02971     }
02972     return std::make_pair(left, right);
02973 }
02974
02975 inline std::string trim_copy(const std::string &s) {
02976     auto r = trim(s.data(), s.data() + s.size(), 0, s.size());
02977     return s.substr(r.first, r.second - r.first);
02978 }
02979

```

```
02980 inline std::string trim_double_quotes_copy(const std::string &s) {
02981     if (s.length() >= 2 && s.front() == '\n' && s.back() == '\n') {
02982         return s.substr(1, s.size() - 2);
02983     }
02984     return s;
02985 }
02986
02987 inline void
02988 divide(const char *data, std::size_t size, char d,
02989         std::function<void(const char *, std::size_t, const char *, std::size_t)>
02990         fn) {
02991     const auto it = std::find(data, data + size, d);
02992     const auto found = static_cast<std::size_t>(it != data + size);
02993     const auto lhs_data = data;
02994     const auto lhs_size = static_cast<std::size_t>(it - data);
02995     const auto rhs_data = it + found;
02996     const auto rhs_size = size - lhs_size - found;
02997
02998     fn(lhs_data, lhs_size, rhs_data, rhs_size);
02999 }
03000
03001 inline void
03002 divide(const std::string &str, char d,
03003         std::function<void(const char *, std::size_t, const char *, std::size_t)>
03004         fn) {
03005     divide(str.data(), str.size(), d, std::move(fn));
03006 }
03007
03008 inline void split(const char *b, const char *e, char d,
03009                      std::function<void(const char *, const char *)> fn) {
03010     return split(b, e, d, (std::numeric_limits<size_t>::max)(), std::move(fn));
03011 }
03012
03013 inline void split(const char *b, const char *e, char d, size_t m,
03014                      std::function<void(const char *, const char *)> fn) {
03015     size_t i = 0;
03016     size_t beg = 0;
03017     size_t count = 1;
03018
03019     while (e ? (b + i < e) : (b[i] != '\0')) {
03020         if (b[i] == d && count < m) {
03021             auto r = trim(b, e, beg, i);
03022             if (r.first < r.second) { fn(&b[r.first], &b[r.second]); }
03023             beg = i + 1;
03024             count++;
03025         }
03026         i++;
03027     }
03028
03029     if (i) {
03030         auto r = trim(b, e, beg, i);
03031         if (r.first < r.second) { fn(&b[r.first], &b[r.second]); }
03032     }
03033 }
03034
03035 inline stream_line_reader::stream_line_reader(Stream &strm, char *fixed_buffer,
03036                                                 size_t fixed_buffer_size)
03037     : strm_(strm), fixed_buffer_(fixed_buffer),
03038       fixed_buffer_size_(fixed_buffer_size) {}
03039
03040 inline const char *stream_line_reader::ptr() const {
03041     if (growable_buffer_.empty()) {
03042         return fixed_buffer_;
03043     } else {
03044         return growable_buffer_.data();
03045     }
03046 }
03047
03048 inline size_t stream_line_reader::size() const {
03049     if (growable_buffer_.empty()) {
03050         return fixed_buffer_used_size_;
03051     } else {
03052         return growable_buffer_.size();
03053     }
03055 }
03056
03057 inline bool stream_line_reader::end_with_crlf() const {
03058     auto end = ptr() + size();
03059     return size() >= 2 && end[-2] == '\r' && end[-1] == '\n';
03060 }
03061
03062 inline bool stream_line_reader::getline() {
03063     fixed_buffer_used_size_ = 0;
03064     growable_buffer_.clear();
03065 #ifndef CPPHTTPLIB_ALLOW_LF_AS_LINE_TERMINATOR
03066     char prev_byte = 0;
```

```

03067 #endif
03068
03069 for (size_t i = 0;; i++) {
03070     char byte;
03071     auto n = strm_.read(&byte, 1);
03072
03073     if (n < 0) {
03074         return false;
03075     } else if (n == 0) {
03076         if (i == 0) {
03077             return false;
03078         } else {
03079             break;
03080         }
03081     }
03082     append(byte);
03083
03084 #ifdef CPPHTTPLIB_ALLOW_LF_AS_LINE_TERMINATOR
03085     if (byte == '\n') { break; }
03086 #else
03087     if (prev_byte == '\r' && byte == '\n') { break; }
03088     prev_byte = byte;
03089 #endif
03090 }
03091 }
03092
03093 return true;
03094 }
03095
03096 inline void stream_line_reader::append(char c) {
03097     if (fixed_buffer_used_size_ < fixed_buffer_size_ - 1) {
03098         fixed_buffer_[fixed_buffer_used_size_++] = c;
03099         fixed_buffer_[fixed_buffer_used_size_] = '\0';
03100     } else {
03101         if (growable_buffer_.empty()) {
03102             assert(fixed_buffer_[fixed_buffer_used_size_] == '\0');
03103             growable_buffer_.assign(fixed_buffer_, fixed_buffer_used_size_);
03104         }
03105         growable_buffer_ += c;
03106     }
03107 }
03108
03109 inline mmap::mmap(const char *path) { open(path); }
03110
03111 inline mmap::~mmap() { close(); }
03112
03113 inline bool mmap::open(const char *path) {
03114     close();
03115
03116 #if defined(_WIN32)
03117     auto wpath = u8string_to_wstring(path);
03118     if (wpath.empty()) { return false; }
03119
03120 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03121     hFile_ = ::CreateFile2(wpath.c_str(), GENERIC_READ, FILE_SHARE_READ,
03122                           OPEN_EXISTING, NULL);
03123 #else
03124     hFile_ = ::CreateFileW(wpath.c_str(), GENERIC_READ, FILE_SHARE_READ, NULL,
03125                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
03126 #endif
03127
03128     if (hFile_ == INVALID_HANDLE_VALUE) { return false; }
03129
03130     LARGE_INTEGER size{};
03131     if (!::GetFileSizeEx(hFile_, &size)) { return false; }
03132     // If the following line doesn't compile due to QuadPart, update Windows SDK.
03133     // See:
03134     // https://github.com/yhirose/cpp-httplib/issues/1903#issuecomment-2316520721
03135     if (static_cast<ULONGLONG>(size.QuadPart) >
03136         (std::numeric_limits<decltype(size_)>::max)()) {
03137         // `size_t` might be 32-bits, on 32-bits Windows.
03138         return false;
03139     }
03140     size_ = static_cast<size_t>(size.QuadPart);
03141
03142 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03143     hMapping_ =
03144         ::CreateFileMappingFromApp(hFile_, NULL, PAGE_READONLY, size_, NULL);
03145 #else
03146     hMapping_ = ::CreateFileMappingW(hFile_, NULL, PAGE_READONLY, 0, 0, NULL);
03147 #endif
03148
03149     // Special treatment for an empty file...
03150     if (hMapping_ == NULL && size_ == 0) {
03151         close();
03152         is_open_empty_file_ = true;
03153         return true;

```

```

03154  }
03155
03156  if (hMapping_ == NULL) {
03157    close();
03158    return false;
03159  }
03160
03161 #if _WIN32_WINNT >= _WIN32_WINNT_WIN8
03162  addr_ = ::MapViewOfFileFromApp(hMapping_, FILE_MAP_READ, 0, 0);
03163 #else
03164  addr_ = ::MapViewOfFile(hMapping_, FILE_MAP_READ, 0, 0, 0);
03165 #endif
03166
03167  if (addr_ == nullptr) {
03168    close();
03169    return false;
03170  }
03171 #else
03172  fd_ = ::open(path, O_RDONLY);
03173  if (fd_ == -1) { return false; }
03174
03175  struct stat sb;
03176  if (fstat(fd_, &sb) == -1) {
03177    close();
03178    return false;
03179  }
03180  size_ = static_cast<size_t>(sb.st_size);
03181
03182  addr_ = ::mmap(NULL, size_, PROT_READ, MAP_PRIVATE, fd_, 0);
03183
03184 // Special treatment for an empty file...
03185  if (addr_ == MAP_FAILED && size_ == 0) {
03186    close();
03187    is_open_empty_file = true;
03188    return false;
03189  }
03190 #endif
03191
03192  return true;
03193 }
03194
03195 inline bool mmap::is_open() const {
03196  return is_open_empty_file ? true : addr_ != nullptr;
03197 }
03198
03199 inline size_t mmap::size() const { return size_; }
03200
03201 inline const char *mmap::data() const {
03202  return is_open_empty_file ? "" : static_cast<const char *>(addr_);
03203 }
03204
03205 inline void mmap::close() {
03206 #if defined(_WIN32)
03207  if (addr_) {
03208    ::UnmapViewOfFile(addr_);
03209    addr_ = nullptr;
03210  }
03211
03212  if (hMapping_) {
03213    ::CloseHandle(hMapping_);
03214    hMapping_ = NULL;
03215  }
03216
03217  if (hFile_ != INVALID_HANDLE_VALUE) {
03218    ::CloseHandle(hFile_);
03219    hFile_ = INVALID_HANDLE_VALUE;
03220  }
03221
03222  is_open_empty_file = false;
03223 #else
03224  if (addr_ != nullptr) {
03225    munmap(addr_, size_);
03226    addr_ = nullptr;
03227  }
03228
03229  if (fd_ != -1) {
03230    ::close(fd_);
03231    fd_ = -1;
03232  }
03233 #endif
03234  size_ = 0;
03235 }
03236 inline int close_socket(socket_t sock) {
03237 #ifdef _WIN32
03238  return closesocket(sock);
03239 #else
03240  return close(sock);

```

```

03241 #endif
03242 }
03243
03244 template <typename T> inline ssize_t handle_EINTR(T fn) {
03245     ssize_t res = 0;
03246     while (true) {
03247         res = fn();
03248         if (res < 0 && errno == EINTR) {
03249             std::this_thread::sleep_for(std::chrono::microseconds{1});
03250             continue;
03251         }
03252         break;
03253     }
03254     return res;
03255 }
03256
03257 inline ssize_t read_socket(socket_t sock, void *ptr, size_t size, int flags) {
03258     return handle_EINTR([&]() {
03259         return recv(sock,
03260 #ifdef _WIN32
03261             static_cast<char*>(ptr), static_cast<int>(size),
03262 #else
03263             ptr, size,
03264 #endif
03265             flags);
03266 });
03267 }
03268
03269 inline ssize_t send_socket(socket_t sock, const void *ptr, size_t size,
03270                             int flags) {
03271     return handle_EINTR([&]() {
03272         return send(sock,
03273 #ifdef _WIN32
03274             static_cast<const char*>(ptr), static_cast<int>(size),
03275 #else
03276             ptr, size,
03277 #endif
03278             flags);
03279 });
03280 }
03281
03282 inline int poll_wrapper(struct pollfd *fds, nfds_t nfds, int timeout) {
03283 #ifdef _WIN32
03284     return ::WSAPoll(fds, nfds, timeout);
03285 #else
03286     return ::poll(fds, nfds, timeout);
03287 #endif
03288 }
03289
03290 template <bool Read>
03291 inline ssize_t select_impl(socket_t sock, time_t sec, time_t usec) {
03292     struct pollfd pfd;
03293     pfd.fd = sock;
03294     pfd.events = (Read ? POLLIN : POLLOUT);
03295
03296     auto timeout = static_cast<int>(sec * 1000 + usec / 1000);
03297
03298     return handle_EINTR([&]() { return poll_wrapper(&pfd, 1, timeout); });
03299 }
03300
03301 inline ssize_t select_read(socket_t sock, time_t sec, time_t usec) {
03302     return select_impl<true>(sock, sec, usec);
03303 }
03304
03305 inline ssize_t select_write(socket_t sock, time_t sec, time_t usec) {
03306     return select_impl<false>(sock, sec, usec);
03307 }
03308
03309 inline Error wait_until_socket_is_ready(socket_t sock, time_t sec,
03310                                              time_t usec) {
03311     struct pollfd pfd_read;
03312     pfd_read.fd = sock;
03313     pfd_read.events = POLLIN | POLLOUT;
03314
03315     auto timeout = static_cast<int>(sec * 1000 + usec / 1000);
03316
03317     auto poll_res =
03318         handle_EINTR([&]() { return poll_wrapper(&pfd_read, 1, timeout); });
03319
03320     if (poll_res == 0) { return Error::ConnectionTimeout; }
03321
03322     if (poll_res > 0 && pfd_read.revents & (POLLIN | POLLOUT)) {
03323         auto error = 0;
03324         socklen_t len = sizeof(error);
03325         auto res = getsockopt(sock, SOL_SOCKET, SO_ERROR,
03326                               reinterpret_cast<char*>(&error), &len);
03327         auto successful = res >= 0 && !error;

```

```
03328     return successful ? Error::Success : Error::Connection;
03329 }
03330
03331 return Error::Connection;
03332 }
03333
03334 inline bool is_socket_alive(socket_t sock) {
03335     const auto val = detail::select_read(sock, 0, 0);
03336     if (val == 0) {
03337         return true;
03338     } else if (val < 0 && errno == EBADF) {
03339         return false;
03340     }
03341     char buf[1];
03342     return detail::read_socket(sock, &buf[0], sizeof(buf), MSG_PEEK) > 0;
03343 }
03344
03345 class SocketStream final : public Stream {
03346 public:
03347     SocketStream(socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec,
03348                  time_t write_timeout_sec, time_t write_timeout_usec,
03349                  time_t max_timeout_msec = 0,
03350                  std::chrono::time_point<std::chrono::steady_clock> start_time =
03351                     (std::chrono::steady_clock::time_point::min}());
03352     ~SocketStream() override;
03353
03354     bool is_readable() const override;
03355     bool wait_readable() const override;
03356     bool wait_writable() const override;
03357     ssize_t read(char *ptr, size_t size) override;
03358     ssize_t write(const char *ptr, size_t size) override;
03359     void get_remote_ip_and_port(std::string &ip, int &port) const override;
03360     void get_local_ip_and_port(std::string &ip, int &port) const override;
03361     socket_t socket() const override;
03362     time_t duration() const override;
03363
03364 private:
03365     socket_t sock_;
03366     time_t read_timeout_sec_;
03367     time_t read_timeout_usec_;
03368     time_t write_timeout_sec_;
03369     time_t write_timeout_usec_;
03370     time_t max_timeout_msec_;
03371     const std::chrono::time_point<std::chrono::steady_clock> start_time_;
03372
03373     std::vector<char> read_buff_;
03374     size_t read_buff_off_ = 0;
03375     size_t read_buff_content_size_ = 0;
03376
03377     static const size_t read_buff_size_ = 1024l * 4;
03378 };
03379
03380 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
03381 class SSLSocketStream final : public Stream {
03382 public:
03383     SSLSocketStream(
03384         socket_t sock, SSL *ssl, time_t read_timeout_sec,
03385         time_t read_timeout_usec, time_t write_timeout_sec,
03386         time_t write_timeout_usec, time_t max_timeout_msec = 0,
03387         std::chrono::time_point<std::chrono::steady_clock> start_time =
03388             (std::chrono::steady_clock::time_point::min}());
03389     ~SSLSocketStream() override;
03390
03391     bool is_readable() const override;
03392     bool wait_readable() const override;
03393     bool wait_writable() const override;
03394     ssize_t read(char *ptr, size_t size) override;
03395     ssize_t write(const char *ptr, size_t size) override;
03396     void get_remote_ip_and_port(std::string &ip, int &port) const override;
03397     void get_local_ip_and_port(std::string &ip, int &port) const override;
03398     socket_t socket() const override;
03399     time_t duration() const override;
03400
03401 private:
03402     socket_t sock_;
03403     SSL *ssl_;
03404     time_t read_timeout_sec_;
03405     time_t read_timeout_usec_;
03406     time_t write_timeout_sec_;
03407     time_t write_timeout_usec_;
03408     time_t max_timeout_msec_;
03409     const std::chrono::time_point<std::chrono::steady_clock> start_time_;
03410 };
03411 #endif
03412
03413 inline bool keep_alive(const std::atomic<socket_t> &svr_sock, socket_t sock,
03414                         time_t keep_alive_timeout_sec) {
```

```
03415 using namespace std::chrono;
03416
03417 const auto interval_usec =
03418     CPPHTTPPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL_USECOND;
03419
03420 // Avoid expensive `steady_clock::now()` call for the first time
03421 if (select_read(sock, 0, interval_usec) > 0) { return true; }
03422
03423 const auto start = steady_clock::now() - microseconds{interval_usec};
03424 const auto timeout = seconds{keep_alive_timeout_sec};
03425
03426 while (true) {
03427     if (svr_sock == INVALID_SOCKET) {
03428         break; // Server socket is closed
03429     }
03430
03431     auto val = select_read(sock, 0, interval_usec);
03432     if (val < 0) {
03433         break; // Socket error
03434     } else if (val == 0) {
03435         if (steady_clock::now() - start > timeout) {
03436             break; // Timeout
03437         }
03438     } else {
03439         return true; // Ready for read
03440     }
03441 }
03442
03443 return false;
03444 }
03445
03446 template <typename T>
03447 inline bool
03448 process_server_socket_core(const std::atomic<socket_t> &svr_sock, socket_t sock,
03449                             size_t keep_alive_max_count,
03450                             time_t keep_alive_timeout_sec, T callback) {
03451     assert(keep_alive_max_count > 0);
03452     auto ret = false;
03453     auto count = keep_alive_max_count;
03454     while (count > 0 && keep_alive(svr_sock, sock, keep_alive_timeout_sec)) {
03455         auto close_connection = count == 1;
03456         auto connection_closed = false;
03457         ret = callback(close_connection, connection_closed);
03458         if (!ret || connection_closed) { break; }
03459         count--;
03460     }
03461     return ret;
03462 }
03463
03464 template <typename T>
03465 inline bool
03466 process_server_socket(const std::atomic<socket_t> &svr_sock, socket_t sock,
03467                         size_t keep_alive_max_count,
03468                         time_t keep_alive_timeout_sec, time_t read_timeout_sec,
03469                         time_t read_timeout_usec, time_t write_timeout_sec,
03470                         time_t write_timeout_usec, T callback) {
03471     return process_server_socket_core(
03472         svr_sock, sock, keep_alive_max_count, keep_alive_timeout_sec,
03473         [&](bool close_connection, bool &connection_closed) {
03474             SocketStream strm(sock, read_timeout_sec, read_timeout_usec,
03475                               write_timeout_sec, write_timeout_usec);
03476             return callback(strm, close_connection, connection_closed);
03477         });
03478 }
03479
03480 inline bool process_client_socket(
03481     socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec,
03482     time_t write_timeout_sec, time_t write_timeout_usec,
03483     time_t max_timeout_msec,
03484     std::chrono::time_point<std::chrono::steady_clock> start_time,
03485     std::function<bool(Stream &)> callback) {
03486     SocketStream strm(sock, read_timeout_sec, read_timeout_usec,
03487                       write_timeout_sec, write_timeout_usec, max_timeout_msec,
03488                       start_time);
03489     return callback(strm);
03490 }
03491
03492 inline int shutdown_socket(socket_t sock) {
03493 #ifdef _WIN32
03494     return shutdown(sock, SD_BOTH);
03495 #else
03496     return shutdown(sock, SHUT_RDWR);
03497 #endif
03498 }
03499
03500 inline std::string escape_abstract_namespace_unix_domain(const std::string &s) {
03501     if (s.size() > 1 && s[0] == '\0') {
```

```

03502     auto ret = s;
03503     ret[0] = '@';
03504     return ret;
03505   }
03506   return s;
03507 }
03508
03509 inline std::string
03510 unescape_abstract_namespace_unix_domain(const std::string &s) {
03511   if (s.size() > 1 && s[0] == '@') {
03512     auto ret = s;
03513     ret[0] = '\0';
03514     return ret;
03515   }
03516   return s;
03517 }
03518
03519 template <typename BindOrConnect>
03520 socket_t create_socket(const std::string &host, const std::string &ip, int port,
03521                         int address_family, int socket_flags, bool tcp_nodelay,
03522                         bool ipv6_v6only, SocketOptions socket_options,
03523                         BindOrConnect bind_or_connect) {
03524   // Get address info
03525   const char *node = nullptr;
03526   struct addrinfo hints;
03527   struct addrinfo *result;
03528
03529   memset(&hints, 0, sizeof(struct addrinfo));
03530   hints.ai_socktype = SOCK_STREAM;
03531   hints.ai_protocol = IPPROTO_IP;
03532
03533   if (!ip.empty()) {
03534     node = ip.c_str();
03535     // Ask getaddrinfo to convert IP in c-string to address
03536     hints.ai_family = AF_UNSPEC;
03537     hints.ai_flags = AI_NUMERICHOST;
03538   } else {
03539     if (!host.empty()) { node = host.c_str(); }
03540     hints.ai_family = address_family;
03541     hints.ai_flags = socket_flags;
03542   }
03543
03544   if (hints.ai_family == AF_UNIX) {
03545     const auto addrlen = host.length();
03546     if (addrlen > sizeof(sockaddr_un::sun_path)) { return INVALID_SOCKET; }
03547
03548 #ifdef SOCK_CLOEXEC
03549   auto sock = socket(hints.ai_family, hints.ai_socktype | SOCK_CLOEXEC,
03550                      hints.ai_protocol);
03551 #else
03552   auto sock = socket(hints.ai_family, hints.ai_socktype, hints.ai_protocol);
03553 #endif
03554
03555   if (sock != INVALID_SOCKET) {
03556     sockaddr_un addr{};
03557     addr.sun_family = AF_UNIX;
03558
03559     auto unescaped_host = unescape_abstract_namespace_unix_domain(host);
03560     std::copy(unescaped_host.begin(), unescaped_host.end(), addr.sun_path);
03561
03562     hints.ai_addr = reinterpret_cast<sockaddr*>(&addr);
03563     hints.ai_addrlen = static_cast<socklen_t>(
03564       sizeof(addr) - sizeof(addr.sun_path) + addrlen);
03565
03566 #ifndef SOCK_CLOEXEC
03567 #ifndef WIN32
03568   fcntl(sock, F_SETFD, FD_CLOEXEC);
03569 #endif
03570 #endif
03571
03572   if (socket_options) { socket_options(sock); }
03573
03574 #ifdef WIN32
03575   // Setting SO_REUSEADDR seems not to work well with AF_UNIX on windows, so
03576   // remove the option.
03577   detail::set_socket_opt(sock, SOL_SOCKET, SO_REUSEADDR, 0);
03578 #endif
03579
03580   bool dummy;
03581   if (!bind_or_connect(sock, hints, dummy)) {
03582     close_socket(sock);
03583     sock = INVALID_SOCKET;
03584   }
03585 }
03586 return sock;
03587 }
03588

```

```
03589 auto service = std::to_string(port);
03590
03591 if (getaddrinfo(node, service.c_str(), &hints, &result)) {
03592 #if defined __linux__ && !defined __ANDROID__
03593     res_init();
03594 #endif
03595     return INVALID_SOCKET;
03596 }
03597 auto se = detail::scope_exit([&] { freeaddrinfo(result); });
03598
03599 for (auto rp = result; rp; rp = rp->ai_next) {
03600     // Create a socket
03601 #ifdef _WIN32
03602     auto sock =
03603         WSASocketW(rp->ai_family, rp->ai_socktype, rp->ai_protocol, nullptr, 0,
03604                     WSA_FLAG_NO_HANDLE_INHERIT | WSA_FLAG_OVERLAPPED);
03619     if (sock == INVALID_SOCKET) {
03620         sock = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
03621     }
03622 #else
03623
03624 #ifdef SOCK_CLOEXEC
03625     auto sock =
03626         socket(rp->ai_family, rp->ai_socktype | SOCK_CLOEXEC, rp->ai_protocol);
03627 #else
03628     auto sock = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
03629 #endif
03630
03631 #endif
03632     if (sock == INVALID_SOCKET) { continue; }
03633
03634 #if !defined _WIN32 && !defined SOCK_CLOEXEC
03635     if (fcntl(sock, F_SETFD, FD_CLOEXEC) == -1) {
03636         close_socket(sock);
03637         continue;
03638     }
03639 #endif
03640
03641     if (tcp_nodelay) { set_socket_opt(sock, IPPROTO_TCP, TCP_NODELAY, 1); }
03642
03643     if (rp->ai_family == AF_INET6) {
03644         set_socket_opt(sock, IPPROTO_IPV6, IPV6_V6ONLY, ipv6_v6only ? 1 : 0);
03645     }
03646
03647     if (socket_options) { socket_options(sock); }
03648
03649     // bind or connect
03650     auto quit = false;
03651     if (bind_or_connect(sock, *rp, quit)) { return sock; }
03652
03653     close_socket(sock);
03654
03655     if (quit) { break; }
03656 }
03657
03658 return INVALID_SOCKET;
03659 }
03660
03661 inline void set_nonblocking(socket_t sock, bool nonblocking) {
03662 #ifdef _WIN32
03663     auto flags = nonblocking ? 1UL : 0UL;
03664     ioctlsocket(sock, FIONBIO, &flags);
03665 #else
03666     auto flags = fcntl(sock, F_GETFL, 0);
03667     fcntl(sock, F_SETFL,
03668         nonblocking ? (flags | O_NONBLOCK) : (flags & (~O_NONBLOCK)));
03669 #endif
03670 }
03671
03672 inline bool is_connection_error() {
03673 #ifdef _WIN32
03674     return WSAGetLastError() != WSAEWOULDBLOCK;
03675 #else
03676     return errno != EINPROGRESS;
03677 #endif
03678 }
03679
03680 inline bool bind_ip_address(socket_t sock, const std::string &host) {
03681     struct addrinfo hints;
03682     struct addrinfo *result;
03683
03684     memset(&hints, 0, sizeof(struct addrinfo));
03685     hints.ai_family = AF_UNSPEC;
03686     hints.ai_socktype = SOCK_STREAM;
03687     hints.ai_protocol = 0;
03688
03689     if (getaddrinfo(host.c_str(), "0", &hints, &result)) { return false; }
```

```

03690 auto se = detail::scope_exit([&] { freeaddrinfo(result); });
03691
03692 auto ret = false;
03693 for (auto rp = result; rp; rp = rp->ai_next) {
03694     const auto &ai = *rp;
03695     if (lbind(sock, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen))) {
03696         ret = true;
03697         break;
03698     }
03699 }
03700
03701 return ret;
03702 }
03703
03704 #if !defined _WIN32 && !defined ANDROID && !defined _AIX && !defined __MVS__
03705 #define USE_IF2IP
03706 #endif
03707
03708 #ifdef USE_IF2IP
03709 inline std::string if2ip(int address_family, const std::string &ifn) {
03710     struct ifaddrs *ifap;
03711     getifaddrs(&ifap);
03712     auto se = detail::scope_exit([&] { freeifaddrs(ifap); });
03713
03714     std::string addr_candidate;
03715     for (auto ifa = ifap; ifa; ifa = ifa->ifa_next) {
03716         if (ifa->ifa_addr && ifn == ifa->ifa_name &&
03717             (AF_UNSPEC == address_family ||
03718              ifa->ifa_addr->sa_family == address_family)) {
03719             if (ifa->ifa_addr->sa_family == AF_INET) {
03720                 auto sa = reinterpret_cast<struct sockaddr_in *>(ifa->ifa_addr);
03721                 char buf[INET_ADDRSTRLEN];
03722                 if (inet_ntop(AF_INET, &sa->sin_addr, buf, INET_ADDRSTRLEN)) {
03723                     return std::string(buf, INET_ADDRSTRLEN);
03724                 }
03725             } else if (ifa->ifa_addr->sa_family == AF_INET6) {
03726                 auto sa = reinterpret_cast<struct sockaddr_in6 *>(ifa->ifa_addr);
03727                 if (!IN6_IS_ADDR_LINKLOCAL(&sa->sin6_addr)) {
03728                     char buf[INET6_ADDRSTRLEN] = {};
03729                     if (inet_ntop(AF_INET6, &sa->sin6_addr, buf, INET6_ADDRSTRLEN)) {
03730                         // equivalent to mac's IN6_IS_ADDR_UNIQUE_LOCAL
03731                         auto s6_addr_head = sa->sin6_addr.s6_addr[0];
03732                         if (s6_addr_head == 0xfc || s6_addr_head == 0xfd) {
03733                             addr_candidate = std::string(buf, INET6_ADDRSTRLEN);
03734                         } else {
03735                             return std::string(buf, INET6_ADDRSTRLEN);
03736                         }
03737                     }
03738                 }
03739             }
03740         }
03741     }
03742     return addr_candidate;
03743 }
03744 #endif
03745
03746 inline socket_t create_client_socket(
03747     const std::string &host, const std::string &ip, int port,
03748     int address_family, bool tcp_nodelay, bool ipv6_v6only,
03749     SocketOptions socket_options, time_t connection_timeout_sec,
03750     time_t connection_timeout_usec, time_t read_timeout_sec,
03751     time_t read_timeout_usec, time_t write_timeout_sec,
03752     time_t write_timeout_usec, const std::string &intf, Error &error) {
03753     auto sock = create_socket(
03754         host, ip, port, address_family, 0, tcp_nodelay, ipv6_v6only,
03755         std::move(socket_options),
03756         [&](socket_t sock2, struct addrinfo &ai, bool &quit) -> bool {
03757             if (!intf.empty()) {
03758 #ifdef USE_IF2IP
03759                 auto ip_from_if = if2ip(address_family, intf);
03760                 if (ip_from_if.empty()) { ip_from_if = intf; }
03761                 if (!bind_ip_address(sock2, ip_from_if)) {
03762                     error = Error::BindIPAddress;
03763                     return false;
03764                 }
03765 #endif
03766             }
03767
03768             set_nonblocking(sock2, true);
03769
03770             auto ret =
03771                 ::connect(sock2, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen));
03772
03773             if (ret < 0) {
03774                 if (is_connection_error()) {
03775                     error = Error::Connection;
03776                     return false;
03777             }
03778         }
03779     });
03780
03781     if (error) {
03782         if (is_connection_error()) {
03783             error = Error::Connection;
03784         }
03785     }
03786 }
03787
03788 #endif

```

```

03777     }
03778     error = wait_until_socket_is_ready(sock2, connection_timeout_sec,
03779                                         connection_timeout_usec);
03780     if (error != Error::Success) {
03781         if (error == Error::ConnectionTimeout) { quit = true; }
03782         return false;
03783     }
03784 }
03785
03786     set_nonblocking(sock2, false);
03787     set_socket_opt_time(sock2, SOL_SOCKET, SO_RCVTIMEO, read_timeout_sec,
03788                          read_timeout_usec);
03789     set_socket_opt_time(sock2, SOL_SOCKET, SO_SNDTIMEO, write_timeout_sec,
03790                          write_timeout_usec);
03791
03792     error = Error::Success;
03793     return true;
03794 });
03795
03796 if (sock != INVALID_SOCKET) {
03797     error = Error::Success;
03798 } else {
03799     if (error == Error::Success) { error = Error::Connection; }
03800 }
03801
03802 return sock;
03803 }
03804
03805 inline bool get_ip_and_port(const struct sockaddr_storage &addr,
03806                               socklen_t addr_len, std::string &ip, int &port) {
03807     if (addr.ss_family == AF_INET) {
03808         port = ntohs(reinterpret_cast<const struct sockaddr_in*>(&addr)->sin_port);
03809     } else if (addr.ss_family == AF_INET6) {
03810         port =
03811             ntohs(reinterpret_cast<const struct sockaddr_in6*>(&addr)->sin6_port);
03812     } else {
03813         return false;
03814     }
03815
03816     std::array<char, NI_MAXHOST> ipstr{};
03817     if (getnameinfo(reinterpret_cast<const struct sockaddr*>(&addr), addr_len,
03818                     ipstr.data(), static_cast<socklen_t>(ipstr.size()), nullptr,
03819                     0, NI_NUMERICHOST)) {
03820         return false;
03821     }
03822
03823     ip = ipstr.data();
03824     return true;
03825 }
03826
03827 inline void get_local_ip_and_port(socket_t sock, std::string &ip, int &port) {
03828     struct sockaddr_storage addr;
03829     socklen_t addr_len = sizeof(addr);
03830     if (!getsockname(sock, reinterpret_cast<struct sockaddr*>(&addr),
03831                      &addr_len)) {
03832         get_ip_and_port(addr, addr_len, ip, port);
03833     }
03834 }
03835
03836 inline void get_remote_ip_and_port(socket_t sock, std::string &ip, int &port) {
03837     struct sockaddr_storage addr;
03838     socklen_t addr_len = sizeof(addr);
03839
03840     if (!getpeername(sock, reinterpret_cast<struct sockaddr*>(&addr),
03841                      &addr_len)) {
03842 #ifndef WIN32
03843     if (addr.ss_family == AF_UNIX) {
03844 #if defined(__linux__)
03845         struct ucred ucred;
03846         socklen_t len = sizeof(ucred);
03847         if (getsockopt(sock, SOL_SOCKET, SO_PEERCREDS, &ucred, &len) == 0) {
03848             port = ucred.pid;
03849         }
03850 #elif defined(SOL_LOCAL) && defined(SO_PEERPID) // __APPLE__
03851         pid_t pid;
03852         socklen_t len = sizeof(pid);
03853         if (getsockopt(sock, SOL_LOCAL, SO_PEERPID, &pid, &len) == 0) {
03854             port = pid;
03855         }
03856 #endif
03857         return;
03858     }
03859 #endif
03860     get_ip_and_port(addr, addr_len, ip, port);
03861 }
03862 }
03863

```

```
03864 inline constexpr unsigned int str2tag_core(const char *s, size_t l,
03865                                     unsigned int h) {
03866     return (l == 0)
03867     ? h
03868     : str2tag_core(
03869         s + 1, l - 1,
03870         // Unsets the 6 high bits of h, therefore no overflow happens
03871         (((std::numeric_limits<unsigned int>::max)() >> 6) &
03872         h * 33) ^
03873         static_cast<unsigned char>(*s));
03874 }
03875
03876 inline unsigned int str2tag(const std::string &s) {
03877     return str2tag_core(s.data(), s.size(), 0);
03878 }
03879
03880 namespace udl {
03881
03882 inline constexpr unsigned int operator""_t(const char *s, size_t l) {
03883     return str2tag_core(s, l, 0);
03884 }
03885
03886 } // namespace udl
03887
03888 inline std::string
03889 find_content_type(const std::string &path,
03890                      const std::map<std::string, std::string> &user_data,
03891                      const std::string &default_content_type) {
03892     auto ext = file_extension(path);
03893
03894     auto it = user_data.find(ext);
03895     if (it != user_data.end()) { return it->second; }
03896
03897     using udl::operator""_t;
03898
03899     switch (str2tag(ext)) {
03900     default: return default_content_type;
03901
03902     case "css"_t: return "text/css";
03903     case "csv"_t: return "text/csv";
03904     case "htm"_t:
03905     case "html"_t: return "text/html";
03906     case "js"_t:
03907     case "mjs"_t: return "text/javascript";
03908     case "txt"_t: return "text/plain";
03909     case "vtt"_t: return "text/vtt";
03910
03911     case "apng"_t: return "image/apng";
03912     case "avif"_t: return "image/avif";
03913     case "bmp"_t: return "image/bmp";
03914     case "gif"_t: return "image/gif";
03915     case "png"_t: return "image/png";
03916     case "svg"_t: return "image/svg+xml";
03917     case "webp"_t: return "image/webp";
03918     case "ico"_t: return "image/x-icon";
03919     case "tif"_t: return "image/tiff";
03920     case "tiff"_t: return "image/tiff";
03921     case "jpg"_t:
03922     case "jpeg"_t: return "image/jpeg";
03923
03924     case "mp4"_t: return "video/mp4";
03925     case "mpeg"_t: return "video/mpeg";
03926     case "webm"_t: return "video/webm";
03927
03928     case "mp3"_t: return "audio/mp3";
03929     case "mpga"_t: return "audio/mpeg";
03930     case "weba"_t: return "audio/webm";
03931     case "wav"_t: return "audio/wave";
03932
03933     case "otf"_t: return "font/otf";
03934     case "ttf"_t: return "font/ttf";
03935     case "woff"_t: return "font/woff";
03936     case "woff2"_t: return "font/woff2";
03937
03938     case "7z"_t: return "application/x-7z-compressed";
03939     case "atom"_t: return "application/atom+xml";
03940     case "pdf"_t: return "application/pdf";
03941     case "json"_t: return "application/json";
03942     case "rss"_t: return "application/rss+xml";
03943     case "tar"_t: return "application/x-tar";
03944     case "xht"_t:
03945     case "xhtml"_t: return "application/xhtml+xml";
03946     case "xslt"_t: return "application/xslt+xml";
03947     case "xml"_t: return "application/xml";
03948     case "gz"_t: return "application/gzip";
03949     case "zip"_t: return "application/zip";
03950     case "wasm"_t: return "application/wasm";
```

```

03951 }
03952 }
03953
03954 inline bool can_compress_content_type(const std::string &content_type) {
03955     using udl::operator""_t;
03956
03957     auto tag = str2tag(content_type);
03958
03959     switch (tag) {
03960         case "image/svg+xml"_t:
03961         case "application/javascript"_t:
03962         case "application/json"_t:
03963         case "application/xml"_t:
03964         case "application/protobuf"_t:
03965         case "application/xhtml+xml"_t: return true;
03966
03967     case "text/event-stream"_t: return false;
03968
03969     default: return !content_type.find("text/", 0);
03970 }
03971 }
03972
03973 inline EncodingType encoding_type(const Request &req, const Response &res) {
03974     auto ret =
03975         detail::can_compress_content_type(res.get_header_value("Content-Type"));
03976     if (!ret) { return EncodingType::None; }
03977
03978     const auto &s = req.get_header_value("Accept-Encoding");
03979     (void)(s);
03980
03981 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
03982 // TODO: 'Accept-Encoding' has br, not br;q=0
03983     ret = s.find("br") != std::string::npos;
03984     if (ret) { return EncodingType::Brotli; }
03985 #endif
03986
03987 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
03988 // TODO: 'Accept-Encoding' has gzip, not gzip;q=0
03989     ret = s.find("gzip") != std::string::npos;
03990     if (ret) { return EncodingType::Gzip; }
03991 #endif
03992
03993 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
03994 // TODO: 'Accept-Encoding' has zstd, not zstd;q=0
03995     ret = s.find("zstd") != std::string::npos;
03996     if (ret) { return EncodingType::Zstd; }
03997 #endif
03998
03999     return EncodingType::None;
04000 }
04001
04002 inline bool nocompressor::compress(const char *data, size_t data_length,
04003                                     bool /*last*/, Callback callback) {
04004     if (!data_length) { return true; }
04005     return callback(data, data_length);
04006 }
04007
04008 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
04009 inline gzip_compressor::gzip_compressor() {
04010     std::memset(&strm_, 0, sizeof(strm_));
04011     strm_.zalloc = Z_NULL;
04012     strm_.zfree = Z_NULL;
04013     strm_.opaque = Z_NULL;
04014
04015     is_valid_ = deflateInit2(&strm_, Z_DEFAULT_COMPRESSION, Z_DEFLATED, 31, 8,
04016                             Z_DEFAULT_STRATEGY) == Z_OK;
04017 }
04018
04019 inline gzip_compressor::~gzip_compressor() { deflateEnd(&strm_); }
04020
04021 inline bool gzip_compressor::compress(const char *data, size_t data_length,
04022                                         bool last, Callback callback) {
04023     assert(is_valid_);
04024
04025     do {
04026         constexpr size_t max_avail_in =
04027             (std::numeric_limits<decltype(strm_.avail_in)>::max)();
04028
04029         strm_.avail_in = static_cast<decltype(strm_.avail_in)>(
04030             (std::min)(data_length, max_avail_in));
04031         strm_.next_in = const_cast<Bytef*>(reinterpret_cast<const Bytef*>(data));
04032
04033         data_length -= strm_.avail_in;
04034         data += strm_.avail_in;
04035
04036         auto flush = (last && data_length == 0) ? Z_FINISH : Z_NO_FLUSH;
04037         auto ret = Z_OK;

```

```

04038
04039     std::array<char, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04040     do {
04041         strm_.avail_out = static_cast<uInt>(buff.size());
04042         strm_.next_out = reinterpret_cast<Bytef*>(buff.data());
04043
04044         ret = deflate(&strm_, flush);
04045         if (ret == Z_STREAM_ERROR) { return false; }
04046
04047         if (!callback(buff.data(), buff.size() - strm_.avail_out)) {
04048             return false;
04049         }
04050     } while (strm_.avail_out == 0);
04051
04052     assert((flush == Z_FINISH && ret == Z_STREAM_END) ||
04053            (flush == Z_NO_FLUSH && ret == Z_OK));
04054     assert(strm_.avail_in == 0);
04055 } while (data_length > 0);
04056
04057     return true;
04058 }
04059
04060 inline gzip_decompressor::gzip_decompressor() {
04061     std::memset(&strm_, 0, sizeof(strm_));
04062     strm_.zalloc = Z_NULL;
04063     strm_.zfree = Z_NULL;
04064     strm_.opaque = Z_NULL;
04065
04066     // 15 is the value of wbits, which should be at the maximum possible value
04067     // to ensure that any gzip stream can be decoded. The offset of 32 specifies
04068     // that the stream type should be automatically detected either gzip or
04069     // deflate.
04070     is_valid_ = inflateInit2(&strm_, 32 + 15) == Z_OK;
04071 }
04072
04073 inline gzip_decompressor::~gzip_decompressor() { inflateEnd(&strm_); }
04074
04075 inline bool gzip_decompressor::is_valid() const { return is_valid_; }
04076
04077 inline bool gzip_decompressor::decompress(const char *data, size_t data_length,
04078                                              CallBack callback) {
04079     assert(is_valid_);
04080
04081     auto ret = Z_OK;
04082
04083     do {
04084         constexpr size_t max_avail_in =
04085             (std::numeric_limits<decltype(strm_.avail_in)>::max)();
04086
04087         strm_.avail_in = static_cast<decltype(strm_.avail_in)>(
04088             (std::min)(data_length, max_avail_in));
04089         strm_.next_in = const_cast<Bytef*>(reinterpret_cast<const Bytef*>(data));
04090
04091         data_length -= strm_.avail_in;
04092         data += strm_.avail_in;
04093
04094         std::array<char, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04095         while (strm_.avail_in > 0 && ret == Z_OK) {
04096             strm_.avail_out = static_cast<uInt>(buff.size());
04097             strm_.next_out = reinterpret_cast<Bytef*>(buff.data());
04098
04099             ret = inflate(&strm_, Z_NO_FLUSH);
04100
04101             assert(ret != Z_STREAM_ERROR);
04102             switch (ret) {
04103                 case Z_NEED_DICT:
04104                 case Z_DATA_ERROR:
04105                 case Z_MEM_ERROR: inflateEnd(&strm_); return false;
04106             }
04107
04108             if (!callback(buff.data(), buff.size() - strm_.avail_out)) {
04109                 return false;
04110             }
04111         }
04112
04113         if (ret != Z_OK && ret != Z_STREAM_END) { return false; }
04114
04115     } while (data_length > 0);
04116
04117     return true;
04118 }
04119 #endif
0420
0421 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
0422 inline brotli_compressor::brotli_compressor() {
0423     state_ = BrotliEncoderCreateInstance(nullptr, nullptr, nullptr);
0424 }
```

```

04125
04126 inline brotli_compressor::~brotli_compressor() {
04127     BrotliEncoderDestroyInstance(state_);
04128 }
04129
04130 inline bool brotli_compressor::compress(const char *data, size_t data_length,
04131                                     bool last, Callback callback) {
04132     std::array<uint8_t, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04133
04134     auto operation = last ? BROTLI_OPERATION_FINISH : BROTLI_OPERATION_PROCESS;
04135     auto available_in = data_length;
04136     auto next_in = reinterpret_cast<const uint8_t*>(data);
04137
04138     for (;;) {
04139         if (last) {
04140             if (BrotliEncoderIsFinished(state_)) { break; }
04141         } else {
04142             if (!available_in) { break; }
04143         }
04144
04145         auto available_out = buff.size();
04146         auto next_out = buff.data();
04147
04148         if (!BrotliEncoderCompressStream(state_, operation, &available_in, &next_in,
04149                                         &available_out, &next_out, nullptr)) {
04150             return false;
04151         }
04152
04153         auto output_bytes = buff.size() - available_out;
04154         if (output_bytes) {
04155             callback(reinterpret_cast<const char*>(buff.data()), output_bytes);
04156         }
04157     }
04158
04159     return true;
04160 }
04161
04162 inline brotli_decompressor::brotli_decompressor() {
04163     decoder_s = BrotliDecoderCreateInstance(0, 0, 0);
04164     decoder_r = decoder_s ? BROTLI_DECODER_RESULT_NEEDS_MORE_INPUT
04165                           : BROTLI_DECODER_RESULT_ERROR;
04166 }
04167
04168 inline brotli_decompressor::~brotli_decompressor() {
04169     if (decoder_s) { BrotliDecoderDestroyInstance(decoder_s); }
04170 }
04171
04172 inline bool brotli_decompressor::is_valid() const { return decoder_s; }
04173
04174 inline bool brotli_decompressor::decompress(const char *data,
04175                                              size_t data_length,
04176                                              Callback callback) {
04177     if (decoder_r == BROTLI_DECODER_RESULT_SUCCESS ||
04178          decoder_r == BROTLI_DECODER_RESULT_ERROR) {
04179         return 0;
04180     }
04181
04182     auto next_in = reinterpret_cast<const uint8_t*>(data);
04183     size_t avail_in = data_length;
04184     size_t total_out;
04185
04186     decoder_r = BROTLI_DECODER_RESULT_NEEDS_MORE_OUTPUT;
04187
04188     std::array<char, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04189     while (decoder_r == BROTLI_DECODER_RESULT_NEEDS_MORE_OUTPUT) {
04190         char *next_out = buff.data();
04191         size_t avail_out = buff.size();
04192
04193         decoder_r = BrotliDecoderDecompressStream(
04194             decoder_s, &avail_in, &next_in, &avail_out,
04195             reinterpret_cast<uint8_t**>(&next_out), &total_out);
04196
04197         if (decoder_r == BROTLI_DECODER_RESULT_ERROR) { return false; }
04198
04199         if (!callback(buff.data(), buff.size() - avail_out)) { return false; }
04200     }
04201
04202     return decoder_r == BROTLI_DECODER_RESULT_SUCCESS ||
04203           decoder_r == BROTLI_DECODER_RESULT_NEEDS_MORE_INPUT;
04204 }
04205 #endif
04206
04207 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
04208 inline zstd_compressor::zstd_compressor() {
04209     ctx_ = ZSTD_createCCtx();
04210     ZSTD_CCtx_setParameter(ctx_, ZSTD_c_compressionLevel, ZSTD_fast);
04211 }

```

```

04212
04213 inline zstd_compressor::~zstd_compressor() { ZSTD_freeCCtx(ctx_); }
04214
04215 inline bool zstd_compressor::compress(const char *data, size_t data_length,
04216                                     bool last, Callback callback) {
04217     std::array<char, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04218
04219     ZSTD_EndDirective mode = last ? ZSTD_e_end : ZSTD_e_continue;
04220     ZSTD_inBuffer input = {data, data_length, 0};
04221
04222     bool finished;
04223     do {
04224         ZSTD_outBuffer output = {buff.data(), CPPHTTPLIB_COMPRESSION_BUFSIZ, 0};
04225         size_t const remaining = ZSTD_compressStream2(ctx_, &output, &input, mode);
04226
04227         if (ZSTD_isError(remaining)) { return false; }
04228
04229         if (!callback(buff.data(), output.pos)) { return false; }
04230
04231         finished = last ? (remaining == 0) : (input.pos == input.size);
04232
04233     } while (!finished);
04234
04235     return true;
04236 }
04237
04238 inline zstd_decompressor::zstd_decompressor() { ctx_ = ZSTD_createDCtx(); }
04239
04240 inline zstd_decompressor::~zstd_decompressor() { ZSTD_freeDCtx(ctx_); }
04241
04242 inline bool zstd_decompressor::is_valid() const { return ctx_ != nullptr; }
04243
04244 inline bool zstd_decompressor::decompress(const char *data, size_t data_length,
04245                                         Callback callback) {
04246     std::array<char, CPPHTTPLIB_COMPRESSION_BUFSIZ> buff{};
04247     ZSTD_inBuffer input = {data, data_length, 0};
04248
04249     while (input.pos < input.size) {
04250         ZSTD_outBuffer output = {buff.data(), CPPHTTPLIB_COMPRESSION_BUFSIZ, 0};
04251         size_t const remaining = ZSTD_decompressStream(ctx_, &output, &input);
04252
04253         if (ZSTD_isError(remaining)) { return false; }
04254
04255         if (!callback(buff.data(), output.pos)) { return false; }
04256     }
04257
04258     return true;
04259 }
04260 #endif
04261
04262 inline bool has_header(const Headers &headers, const std::string &key) {
04263     return headers.find(key) != headers.end();
04264 }
04265
04266 inline const char *get_header_value(const Headers &headers,
04267                                       const std::string &key, const char *def,
04268                                       size_t id) {
04269     auto rng = headers.equal_range(key);
04270     auto it = rng.first;
04271     std::advance(it, static_cast<ssize_t>(id));
04272     if (it != rng.second) { return it->second.c_str(); }
04273     return def;
04274 }
04275
04276 template <typename T>
04277 inline bool parse_header(const char *beg, const char *end, T fn) {
04278     // Skip trailing spaces and tabs.
04279     while (beg < end && is_space_or_tab(end[-1])) {
04280         end--;
04281     }
04282
04283     auto p = beg;
04284     while (p < end && *p != ':') {
04285         p++;
04286     }
04287
04288     auto name = std::string(beg, p);
04289     if (!detail::fields::is_field_name(name)) { return false; }
04290
04291     if (p == end) { return false; }
04292
04293     auto key_end = p;
04294
04295     if (*p++ != ':') { return false; }
04296
04297     while (p < end && is_space_or_tab(*p)) {
04298         p++;

```

```
04299  }
04300
04301 if (p <= end) {
04302     auto key_len = key_end - beg;
04303     if (!key_len) { return false; }
04304
04305     auto key = std::string(beg, key_end);
04306     auto val = std::string(p, end);
04307
04308     if (!detail::fields::is_field_value(val)) { return false; }
04309
04310     if (case_ignore::equal(key, "Location") ||
04311         case_ignore::equal(key, "Referer")) {
04312         fn(key, val);
04313     } else {
04314         fn(key, decode_url(val, false));
04315     }
04316
04317     return true;
04318 }
04319
04320 return false;
04321 }
04322
04323 inline bool read_headers(Stream &zstrm, Headers &headers) {
04324     const auto bufsiz = 2048;
04325     char buf[bufsiz];
04326     stream_line_reader line_reader(strm, buf, bufsiz);
04327
04328     for (;;) {
04329         if (!line_reader.getline()) { return false; }
04330
04331         // Check if the line ends with CRLF.
04332         auto line_terminator_len = 2;
04333         if (line_reader.end_with_crlf()) {
04334             // Blank line indicates end of headers.
04335             if (line_reader.size() == 2) { break; }
04336         } else {
04337 #ifdef CPPHTTPLIB_ALLOW_LF_AS_LINE_TERMINATOR
04338             // Blank line indicates end of headers.
04339             if (line_reader.size() == 1) { break; }
04340             line_terminator_len = 1;
04341 #else
04342             continue; // Skip invalid line.
04343 #endif
04344     }
04345
04346     if (line_reader.size() > CPPHTTPLIB_HEADER_MAX_LENGTH) { return false; }
04347
04348     // Exclude line terminator
04349     auto end = line_reader.ptr() + line_reader.size() - line_terminator_len;
04350
04351     if (!parse_header(line_reader.ptr(), end,
04352                     [&](const std::string &key, const std::string &val) {
04353                         headers.emplace(key, val);
04354                     })) {
04355         return false;
04356     }
04357 }
04358
04359 return true;
04360 }
04361
04362 inline bool read_content_with_length(Stream &zstrm, uint64_t len,
04363                                         Progress progress,
04364                                         ContentReceiverWithProgress out) {
04365     char buf[CPPHTTPLIB_RECV_BUFSIZ];
04366
04367     uint64_t r = 0;
04368     while (r < len) {
04369         auto read_len = static_cast<size_t>(len - r);
04370         auto n = strm.read(buf, (std::min)(read_len, CPPHTTPLIB_RECV_BUFSIZ));
04371         if (n <= 0) { return false; }
04372
04373         if (!out(buf, static_cast<size_t>(n), r, len)) { return false; }
04374         r += static_cast<uint64_t>(n);
04375
04376         if (progress) {
04377             if (!progress(r, len)) { return false; }
04378         }
04379     }
04380
04381     return true;
04382 }
04383
04384 inline void skip_content_with_length(Stream &zstrm, uint64_t len) {
04385     char buf[CPPHTTPLIB_RECV_BUFSIZ];
```

```
04386     uint64_t r = 0;
04387     while (r < len) {
04388         auto read_len = static_cast<size_t>(len - r);
04389         auto n = strm.read(buf, (std::min)(read_len, CPPHTTPLIB_RECV_BUFSIZ));
04390         if (n <= 0) { return; }
04391         r += static_cast<uint64_t>(n);
04392     }
04393 }
04394
04395 inline bool read_content_without_length(Stream &strm,
04396                                         ContentReceiverWithProgress out) {
04397     char buf[CPPHTTPLIB_RECV_BUFSIZ];
04398     uint64_t r = 0;
04399     for (;;) {
04400         auto n = strm.read(buf, CPPHTTPLIB_RECV_BUFSIZ);
04401         if (n == 0) { return true; }
04402         if (n < 0) { return false; }
04403
04404         if (!out(buf, static_cast<size_t>(n), r, 0)) { return false; }
04405         r += static_cast<uint64_t>(n);
04406     }
04407
04408     return true;
04409 }
04410
04411 template <typename T>
04412 inline bool read_content_chunked(Stream &strm, T &x,
04413                                     ContentReceiverWithProgress out) {
04414     const auto bufsiz = 16;
04415     char buf[bufsiz];
04416
04417     stream_line_reader line_reader(strm, buf, bufsiz);
04418
04419     if (!line_reader.getline()) { return false; }
04420
04421     unsigned long chunk_len;
04422     while (true) {
04423         char *end_ptr;
04424
04425         chunk_len = std::strtoul(line_reader.ptr(), &end_ptr, 16);
04426
04427         if (end_ptr == line_reader.ptr()) { return false; }
04428         if (chunk_len == ULONG_MAX) { return false; }
04429
04430         if (chunk_len == 0) { break; }
04431
04432         if (!read_content_with_length(strm, chunk_len, nullptr, out)) {
04433             return false;
04434         }
04435
04436         if (!line_reader.getline()) { return false; }
04437
04438         if (strcmp(line_reader.ptr(), "\r\n") != 0) { return false; }
04439
04440         if (!line_reader.getline()) { return false; }
04441     }
04442
04443     assert(chunk_len == 0);
04444
04445 // NOTE: In RFC 9112, '7.1 Chunked Transfer Coding' mentions "The chunked
04446 // transfer coding is complete when a chunk with a chunk-size of zero is
04447 // received, possibly followed by a trailer section, and finally terminated by
04448 // an empty line". https://www.rfc-editor.org/rfc/rfc9112.html#section-7.1
04449 //
04450 // In '7.1.3. Decoding Chunked', however, the pseudo-code in the section
04451 // doesn't care for the existence of the final CRLF. In other words, it seems
04452 // to be ok whether the final CRLF exists or not in the chunked data.
04453 // https://www.rfc-editor.org/rfc/rfc9112.html#section-7.1.3
04454 //
04455 // According to the reference code in RFC 9112, cpp-httplib now allows
04456 // chunked transfer coding data without the final CRLF.
04457 if (!line_reader.getline()) { return true; }
04458
04459 while (strcmp(line_reader.ptr(), "\r\n") != 0) {
04460     if (line_reader.size() > CPPHTTPLIB_HEADER_MAX_LENGTH) { return false; }
04461
04462     // Exclude line terminator
04463     constexpr auto line_terminator_len = 2;
04464     auto end = line_reader.ptr() + line_reader.size() - line_terminator_len;
04465
04466     parse_header(line_reader.ptr(), end,
04467                  [&](const std::string &key, const std::string &val) {
04468                      x.headers.emplace(key, val);
04469                  });
04470
04471     if (!line_reader.getline()) { return false; }
04472 }
```

```
04473     return true;
04474 }
04475 }
04476
04477 inline bool is_chunked_transfer_encoding(const Headers &headers) {
04478     return case_ignore::equal(
04479         get_header_value(headers, "Transfer-Encoding", "", 0), "chunked");
04480 }
04481
04482 template <typename T, typename U>
04483 bool prepare_content_receiver(T &x, int &status,
04484     ContentReceiverWithProgress receiver,
04485     bool decompress, U callback) {
04486     if (decompress) {
04487         std::string encoding = x.get_header_value("Content-Encoding");
04488         std::unique_ptr<decompressor> decompressor;
04489
04490         if (encoding == "gzip" || encoding == "deflate") {
04491 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
04492             decompressor = detail::make_unique<gzip_decompressor>();
04493 #else
04494             status = StatusCode::UnsupportedMediaType_415;
04495             return false;
04496 #endif
04497     } else if (encoding.find("br") != std::string::npos) {
04498 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
04499             decompressor = detail::make_unique<brotli_decompressor>();
04500 #else
04501             status = StatusCode::UnsupportedMediaType_415;
04502             return false;
04503 #endif
04504     } else if (encoding == "zstd") {
04505 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
04506         decompressor = detail::make_unique<zstd_decompressor>();
04507 #else
04508         status = StatusCode::UnsupportedMediaType_415;
04509         return false;
04510 #endif
04511     }
04512
04513     if (decompressor) {
04514         if (decompressor->is_valid()) {
04515             ContentReceiverWithProgress out = [&](const char *buf, size_t n,
04516                 uint64_t off, uint64_t len) {
04517                 return decompressor->decompress(buf, n,
04518                     [&](const char *buf2, size_t n2) {
04519                         return receiver(buf2, n2, off, len);
04520                     });
04521             };
04522             return callback(std::move(out));
04523         } else {
04524             status = StatusCode::InternalServerError_500;
04525             return false;
04526         }
04527     }
04528 }
04529
04530 ContentReceiverWithProgress out = [&](const char *buf, size_t n, uint64_t off,
04531                 uint64_t len) {
04532     return receiver(buf, n, off, len);
04533 };
04534 return callback(std::move(out));
04535 }
04536
04537 template <typename T>
04538 bool read_content(Stream &strm, T &x, size_t payload_max_length, int &status,
04539     Progress progress, ContentReceiverWithProgress receiver,
04540     bool decompress) {
04541     return prepare_content_receiver(
04542         x, status, std::move(receiver), decompress,
04543         [&](const ContentReceiverWithProgress &out) {
04544             auto ret = true;
04545             auto exceed_payload_max_length = false;
04546
04547             if (is_chunked_transfer_encoding(x.headers)) {
04548                 ret = read_content_chunked(strm, x, out);
04549             } else if (!has_header(x.headers, "Content-Length")) {
04550                 ret = read_content_without_length(strm, out);
04551             } else {
04552                 auto is_invalid_value = false;
04553                 auto len = get_header_value_u64(
04554                     x.headers, "Content-Length",
04555                     (std::numeric_limits<uint64_t>::max)(), 0, is_invalid_value);
04556
04557                 if (is_invalid_value) {
04558                     ret = false;
04559                 } else if (len > payload_max_length) {
04560
04561
04562
04563
04564
04565
04566
04567
04568
04569
04570
04571
04572
04573
04574
04575
04576
04577
04578
04579
04580
04581
04582
04583
04584
04585
04586
04587
04588
04589
04590
04591
04592
04593
04594
04595
04596
04597
04598
04599
04600
04601
04602
04603
04604
04605
04606
04607
04608
04609
04610
04611
04612
04613
04614
04615
04616
04617
04618
04619
04620
04621
04622
04623
04624
04625
04626
04627
04628
04629
04630
04631
04632
04633
04634
04635
04636
04637
04638
04639
04640
04641
04642
04643
04644
04645
04646
04647
04648
04649
04650
04651
04652
04653
04654
04655
04656
04657
04658
04659
04660
04661
04662
04663
04664
04665
04666
04667
04668
04669
04670
04671
04672
04673
04674
04675
04676
04677
04678
04679
04680
04681
04682
04683
04684
04685
04686
04687
04688
04689
04690
04691
04692
04693
04694
04695
04696
04697
04698
04699
04700
04701
04702
04703
04704
04705
04706
04707
04708
04709
04710
04711
04712
04713
04714
04715
04716
04717
04718
04719
04720
04721
04722
04723
04724
04725
04726
04727
04728
04729
04730
04731
04732
04733
04734
04735
04736
04737
04738
04739
04740
04741
04742
04743
04744
04745
04746
04747
04748
04749
04750
04751
04752
04753
04754
04755
04756
04757
04758
04759
04760
04761
04762
04763
04764
04765
04766
04767
04768
04769
04770
04771
04772
04773
04774
04775
04776
04777
04778
04779
04780
04781
04782
04783
04784
04785
04786
04787
04788
04789
04790
04791
04792
04793
04794
04795
04796
04797
04798
04799
04800
04801
04802
04803
04804
04805
04806
04807
04808
04809
04810
04811
04812
04813
04814
04815
04816
04817
04818
04819
04820
04821
04822
04823
04824
04825
04826
04827
04828
04829
04830
04831
04832
04833
04834
04835
04836
04837
04838
04839
04840
04841
04842
04843
04844
04845
04846
04847
04848
04849
04850
04851
04852
04853
04854
04855
04856
04857
04858
04859
04860
04861
04862
04863
04864
04865
04866
04867
04868
04869
04870
04871
04872
04873
04874
04875
04876
04877
04878
04879
04880
04881
04882
04883
04884
04885
04886
04887
04888
04889
04890
04891
04892
04893
04894
04895
04896
04897
04898
04899
04900
04901
04902
04903
04904
04905
04906
04907
04908
04909
04910
04911
04912
04913
04914
04915
04916
04917
04918
04919
04920
04921
04922
04923
04924
04925
04926
04927
04928
04929
04930
04931
04932
04933
04934
04935
04936
04937
04938
04939
04940
04941
04942
04943
04944
04945
04946
04947
04948
04949
04950
04951
04952
04953
04954
04955
04956
04957
04958
04959
04960
04961
04962
04963
04964
04965
04966
04967
04968
04969
04970
04971
04972
04973
04974
04975
04976
04977
04978
04979
04980
04981
04982
04983
04984
04985
04986
04987
04988
04989
04990
04991
04992
04993
04994
04995
04996
04997
04998
04999
049999
05000
05001
05002
05003
05004
05005
05006
05007
05008
05009
050010
050011
050012
050013
050014
050015
050016
050017
050018
050019
050020
050021
050022
050023
050024
050025
050026
050027
050028
050029
050030
050031
050032
050033
050034
050035
050036
050037
050038
050039
050040
050041
050042
050043
050044
050045
050046
050047
050048
050049
050050
050051
050052
050053
050054
050055
050056
050057
050058
050059
050060
050061
050062
050063
050064
050065
050066
050067
050068
050069
050070
050071
050072
050073
050074
050075
050076
050077
050078
050079
050080
050081
050082
050083
050084
050085
050086
050087
050088
050089
050090
050091
050092
050093
050094
050095
050096
050097
050098
050099
0500100
0500101
0500102
0500103
0500104
0500105
0500106
0500107
0500108
0500109
0500110
0500111
0500112
0500113
0500114
0500115
0500116
0500117
0500118
0500119
0500120
0500121
0500122
0500123
0500124
0500125
0500126
0500127
0500128
0500129
0500130
0500131
0500132
0500133
0500134
0500135
0500136
0500137
0500138
0500139
0500140
0500141
0500142
0500143
0500144
0500145
0500146
0500147
0500148
0500149
0500150
0500151
0500152
0500153
0500154
0500155
0500156
0500157
0500158
0500159
0500160
0500161
0500162
0500163
0500164
0500165
0500166
0500167
0500168
0500169
0500170
0500171
0500172
0500173
0500174
0500175
0500176
0500177
0500178
0500179
0500180
0500181
0500182
0500183
0500184
0500185
0500186
0500187
0500188
0500189
0500190
0500191
0500192
0500193
0500194
0500195
0500196
0500197
0500198
0500199
0500200
0500201
0500202
0500203
0500204
0500205
0500206
0500207
0500208
0500209
0500210
0500211
0500212
0500213
0500214
0500215
0500216
0500217
0500218
0500219
0500220
0500221
0500222
0500223
0500224
0500225
0500226
0500227
0500228
0500229
0500230
0500231
0500232
0500233
0500234
0500235
0500236
0500237
0500238
0500239
0500240
0500241
0500242
0500243
0500244
0500245
0500246
0500247
0500248
0500249
0500250
0500251
0500252
0500253
0500254
0500255
0500256
0500257
0500258
0500259
0500260
0500261
0500262
0500263
0500264
0500265
0500266
0500267
0500268
0500269
0500270
0500271
0500272
0500273
0500274
0500275
0500276
0500277
0500278
0500279
0500280
0500281
0500282
0500283
0500284
0500285
0500286
0500287
0500288
0500289
0500290
0500291
0500292
0500293
0500294
0500295
0500296
0500297
0500298
0500299
0500300
0500301
0500302
0500303
0500304
0500305
0500306
0500307
0500308
0500309
0500310
0500311
0500312
0500313
0500314
0500315
0500316
0500317
0500318
0500319
0500320
0500321
0500322
0500323
0500324
0500325
0500326
0500327
0500328
0500329
0500330
0500331
0500332
0500333
0500334
0500335
0500336
0500337
0500338
0500339
0500340
0500341
0500342
0500343
0500344
0500345
0500346
0500347
0500348
0500349
0500350
0500351
0500352
0500353
0500354
0500355
0500356
0500357
0500358
0500359
0500360
0500361
0500362
0500363
0500364
0500365
0500366
0500367
0500368
0500369
0500370
0500371
0500372
0500373
0500374
0500375
0500376
0500377
0500378
0500379
0500380
0500381
0500382
0500383
0500384
0500385
0500386
0500387
0500388
0500389
0500390
0500391
0500392
0500393
0500394
0500395
0500396
0500397
0500398
0500399
0500400
0500401
0500402
0500403
0500404
0500405
0500406
0500407
0500408
0500409
0500410
0500411
0500412
0500413
0500414
0500415
0500416
0500417
0500418
0500419
0500420
0500421
0500422
0500423
0500424
0500425
0500426
0500427
0500428
0500429
0500430
0500431
0500432
0500433
0500434
0500435
0500436
0500437
0500438
0500439
0500440
0500441
0500442
0500443
0500444
0500445
0500446
0500447
0500448
0500449
0500450
0500451
0500452
0500453
0500454
0500455
0500456
0500457
0500458
0500459
0500460
0500461
0500462
0500463
0500464
0500465
0500466
0500467
0500468
0500469
0500470
0500471
0500472
0500473
0500474
0500475
0500476
0500477
0500478
0500479
0500480
0500481
0500482
0500483
0500484
0500485
0500486
0500487
0500488
0500489
0500490
0500491
0500492
0500493
0500494
0500495
0500496
0500497
0500498
0500499
0500500
0500501
0500502
0500503
0500504
0500505
0500506
0500507
0500508
0500509
0500510
0500511
0500512
0500513
0500514
0500515
0500516
0500517
0500518
0500519
0500520
0500521
0500522
0500523
0500524
0500525
0500526
0500527
0500528
0500529
0500530
0500531
0500532
0500533
0500534
0500535
0500536
0500537
0500538
0500539
0500540
0500541
0500542
0500543
0500544
0500545
0500546
0500547
0500548
0500549
0500550
0500551
0500552
0500553
0500554
0500555
0500556
0500557
0500558
0500559
0500560
0500561
0500562
0500563
0500564
0500565
0500566
0500567
0500568
0500569
0500570
0500571
0500572
0500573
0500574
0500575
0500576
0500577
0500578
0500579
0500580
0500581
0500582
0500583
0500584
0500585
0500586
0500587
0500588
0500589
0500590
0500591
0500592
0500593
0500594
0500595
0500596
0500597
0500598
0500599
0500600
0500601
0500602
0500603
0500604
0500605
0500606
0500607
0500608
0500609
0500610
0500611
0500612
0500613
0500614
0500615
0500616
0500617
0500618
0500619
0500620
0500621
0500622
0500623
0500624
0500625
0500626
0500627
0500628
0500629
0500630
0500631
0500632
0500633
0500634
0500635
0500636
0500637
0500638
0500639
0500640
0500641
0500642
0500643
0500644
0500645
0500646
0500647
0500648
0500649
0500650
0500651
0500652
0500653
0500654
0500655
0500656
0500657
0500658
0500659
0500660
0500661
0500662
0500663
0500664
0500665
0500666
0500667
0500668
0500669
0500670
0500671
0500672
0500673
0500674
0500675
0500676
0500677
0500678
0500679
0500680
0500681
0500682
0500683
0500684
0500685
0500686
0500687
0500688
0500689
0500690
0500691
0500692
0500693
0500694
0500695
0500696
0500697
0500698
0500699
0500700
0500701
0500702
0500703
0500704
0500705
0500706
0500707
0500708
0500709
0500710
0500711
0500712
0500713
0500714
0500715
0500716
0500717
0500718
0500719
0500720
0500721
0500722
0500723
0500724
0500725
0500726
0500727
0500728
0500729
0500730
0500731
0500732
0500733
0500734
0500735
0500736
0500737
0500738
0500739
0500740
0500741
0500742
0500743
0500744
0500745
0500746
0500747
0500748
0500749
0500750
0500751
0500752
0500753
0500754
0500755
0500756
0500757
0500758
0500759
0500760
0500761
0500762
0500763
0500764
0500765
0500766
0500767
0500768
0500769
0500770
0500771
0500772
0500773
0500774
0500775
0500776
0500777
0500778
0500779
0500780
0500781
0500782
0500783
0500784
0500785
0500786
0500787
0500788
0500789
0500790
0500791
0500792
0500793
0500794
0500795
0500796
0500797
0500798
0500799
0500800
0500801
0500802
0500803
0500804
0500805
0500806
0500807
0500808
0500809
0500810
0500811
0500812
0500813
0500814
0500815
0500816
0500817
0500818
0500819
0500820
0500821
0500822
0500823
0500824
0500825
0500826
0500827
0500828
0500829
0500830
0500831
0500832
0500833
0500834
0500835
0500836
0500837
0500838
0500839
0500840
0500841
0500842
0500843
0500844
0500845
0500846
0500847
0500848
0500849
0500850
0500851
0500852
0500853
0500854
0500855
0500856
0500857
0500858
0500859
0500860
0500861
0500862
0500863
0500864
0500865
0500866
0500867
0500868
0500869
0500870
0500871
0500872
0500873
0500874
0500875
0500876
0500877
0500878
0500879
0500880
0500881
0500882
0500883
0500884
0500885
0500886
0500887
0500888
0500889
0500890
0500891
0500892
0500893
0500894
0500895
0500896
0500897
0500898
0500899
0500900
0500901
0500902
0500903
0500904
0500905
0500906
0500907
0500908
0500909
0500910
0500911
0500912
0500913
0500914
0500915
0500916
0500917
0500918
0500919
```

```
04560     exceed_payload_max_length = true;
04561     skip_content_with_length(strm, len);
04562     ret = false;
04563 } else if (len > 0) {
04564     ret = read_content_with_length(strm, len, std::move(progress), out);
04565 }
04566 }
04567
04568 if (!ret) {
04569     status = exceed_payload_max_length ? StatusCode::PayloadTooLarge_413
04570             : StatusCode::BadRequest_400;
04571 }
04572     return ret;
04573 });
04574 }
04575
04576 inline ssize_t write_request_line(Stream &strm, const std::string &method,
04577                                     const std::string &path) {
04578     std::string s = method;
04579     s += " ";
04580     s += path;
04581     s += " HTTP/1.1\r\n";
04582     return strm.write(s.data(), s.size());
04583 }
04584
04585 inline ssize_t write_response_line(Stream &strm, int status) {
04586     std::string s = "HTTP/1.1 ";
04587     s += std::to_string(status);
04588     s += " ";
04589     s += httpplib::status_message(status);
04590     s += "\r\n";
04591     return strm.write(s.data(), s.size());
04592 }
04593
04594 inline ssize_t write_headers(Stream &strm, const Headers &headers) {
04595     ssize_t write_len = 0;
04596     for (const auto &x : headers) {
04597         std::string s;
04598         s = x.first;
04599         s += ": ";
04600         s += x.second;
04601         s += "\r\n";
04602
04603         auto len = strm.write(s.data(), s.size());
04604         if (len < 0) { return len; }
04605         write_len += len;
04606     }
04607     auto len = strm.write("\r\n");
04608     if (len < 0) { return len; }
04609     write_len += len;
04610     return write_len;
04611 }
04612
04613 inline bool write_data(Stream &strm, const char *d, size_t l) {
04614     size_t offset = 0;
04615     while (offset < l) {
04616         auto length = strm.write(d + offset, l - offset);
04617         if (length < 0) { return false; }
04618         offset += static_cast<size_t>(length);
04619     }
04620     return true;
04621 }
04622
04623 template <typename T>
04624 inline bool write_content(Stream &strm, const ContentProvider &content_provider,
04625                               size_t offset, size_t length, bool is_shutting_down,
04626                               Error &error) {
04627     size_t end_offset = offset + length;
04628     auto ok = true;
04629     DataSink data_sink;
04630
04631     data_sink.write = [&](const char *d, size_t l) -> bool {
04632         if (ok) {
04633             if (write_data(strm, d, l)) {
04634                 offset += l;
04635             } else {
04636                 ok = false;
04637             }
04638         }
04639         return ok;
04640     };
04641
04642     data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04643
04644     while (offset < end_offset && !is_shutting_down()) {
04645         if (!strm.wait_writable()) {
04646             error = Error::Write;
```

```

04647     return false;
04648 } else if (!content_provider(offset, end_offset - offset, data_sink)) {
04649     error = Error::Canceled;
04650     return false;
04651 } else if (!ok) {
04652     error = Error::Write;
04653     return false;
04654 }
04655 }
04656
04657 error = Error::Success;
04658 return true;
04659 }
04660
04661 template <typename T>
04662 inline bool write_content(Stream &strm, const ContentProvider &content_provider,
04663             size_t offset, size_t length,
04664             const T &is_shutting_down) {
04665     auto error = Error::Success;
04666     return write_content(strm, content_provider, offset, length, is_shutting_down,
04667                           error);
04668 }
04669
04670 template <typename T>
04671 inline bool
04672 write_content_without_length(Stream &strm,
04673             const ContentProvider &content_provider,
04674             const T &is_shutting_down) {
04675     size_t offset = 0;
04676     auto data_available = true;
04677     auto ok = true;
04678     DataSink data_sink;
04679
04680     data_sink.write = [&](const char *d, size_t l) -> bool {
04681         if (ok) {
04682             offset += l;
04683             if (!write_data(strm, d, l)) { ok = false; }
04684         }
04685         return ok;
04686     };
04687
04688     data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04689
04690     data_sink.done = [&](void) { data_available = false; };
04691
04692     while (data_available && !is_shutting_down()) {
04693         if (!strm.wait_writable()) {
04694             return false;
04695         } else if (!content_provider(offset, 0, data_sink)) {
04696             return false;
04697         } else if (!ok) {
04698             return false;
04699         }
04700     }
04701     return true;
04702 }
04703
04704 template <typename T, typename U>
04705 inline bool
04706 write_content_chunked(Stream &strm, const ContentProvider &content_provider,
04707             const T &is_shutting_down, U &compressor, Error &error) {
04708     size_t offset = 0;
04709     auto data_available = true;
04710     auto ok = true;
04711     DataSink data_sink;
04712
04713     data_sink.write = [&](const char *d, size_t l) -> bool {
04714         if (!ok) {
04715             data_available = l > 0;
04716             offset += l;
04717
04718             std::string payload;
04719             if (compressor.compress(d, l, false,
04720                         [&](const char *data, size_t data_len) {
04721                             payload.append(data, data_len);
04722                             return true;
04723                         })) {
04724                 if (!payload.empty()) {
04725                     // Emit chunked response header and footer for each chunk
04726                     auto chunk =
04727                         from_i_to_hex(payload.size()) + "\r\n" + payload + "\r\n";
04728                     if (!write_data(strm, chunk.data(), chunk.size())) { ok = false; }
04729                 }
04730             } else {
04731                 ok = false;
04732             }
04733         }
04734     };

```

```

04734     return ok;
04735   };
04736 
04737   data_sink.is_writable = [&]() -> bool { return strm.wait_writable(); };
04738 
04739   auto done_with_trailer = [&](const Headers *trailer) {
04740     if (!ok) { return; }
04741 
04742     data_available = false;
04743 
04744     std::string payload;
04745     if (!compressor.compress(nullptr, 0, true,
04746                             [&](const char *data, size_t data_len) {
04747                               payload.append(data, data_len);
04748                               return true;
04749                             })) {
04750       ok = false;
04751       return;
04752     }
04753 
04754     if (!payload.empty()) {
04755       // Emit chunked response header and footer for each chunk
04756       auto chunk = from_i_to_hex(payload.size()) + "\r\n" + payload + "\r\n";
04757       if (!write_data(strm, chunk.data(), chunk.size())) {
04758         ok = false;
04759         return;
04760       }
04761     }
04762 
04763     constexpr const char done_marker[] = "0\r\n";
04764     if (!write_data(strm, done_marker, str_len(done_marker))) { ok = false; }
04765 
04766     // Trailer
04767     if (trailer) {
04768       for (const auto &kv : *trailer) {
04769         std::string field_line = kv.first + ": " + kv.second + "\r\n";
04770         if (!write_data(strm, field_line.data(), field_line.size())) {
04771           ok = false;
04772         }
04773       }
04774     }
04775 
04776     constexpr const char crlf[] = "\r\n";
04777     if (!write_data(strm, crlf, str_len(crlf))) { ok = false; }
04778   };
04779 
04780   data_sink.done = [&](void) { done_with_trailer(nullptr); };
04781 
04782   data_sink.done_with_trailer = [&](const Headers &trailer) {
04783     done_with_trailer(&trailer);
04784   };
04785 
04786   while (data_available && !is_shutting_down()) {
04787     if (!strm.wait_writable()) {
04788       error = Error::Write;
04789       return false;
04790     } else if (!content_provider(offset, 0, data_sink)) {
04791       error = Error::Canceled;
04792       return false;
04793     } else if (!ok) {
04794       error = Error::Write;
04795       return false;
04796     }
04797   }
04798 
04799   error = Error::Success;
04800   return true;
04801 }
04802 
04803 template <typename T, typename U>
04804 inline bool write_content_chunked(Stream &strm,
04805                                     const ContentProvider &content_provider,
04806                                     const T &is_shutting_down, U &compressor) {
04807   auto error = Error::Success;
04808   return write_content_chunked(strm, content_provider, is_shutting_down,
04809                                compressor, error);
04810 }
04811 
04812 template <typename T>
04813 inline bool redirect(T &cli, Request &req, Response &res,
04814                       const std::string &path, const std::string &location,
04815                       Error &error) {
04816   Request new_req = req;
04817   new_req.path = path;
04818   new_req.redirect_count_ -= 1;
04819 
04820   if (res.status == StatusCode::SeeOther_303 &&

```

```
04821     (req.method != "GET" && req.method != "HEAD")) {  
04822     new_req.method = "GET";  
04823     new_req.body.clear();  
04824     new_req.headers.clear();  
04825 }  
04826  
04827 Response new_res;  
04828  
04829 auto ret = cli.send(new_req, new_res, error);  
04830 if (ret) {  
04831     req = new_req;  
04832     res = new_res;  
04833  
04834     if (res.location.empty()) { res.location = location; }  
04835 }  
04836 return ret;  
04837 }  
04838  
04839 inline std::string params_to_query_str(const Params &params) {  
04840     std::string query;  
04841  
04842     for (auto it = params.begin(); it != params.end(); ++it) {  
04843         if (it != params.begin()) { query += "&"; }  
04844         query += it->first;  
04845         query += "=";  
04846         query += encode_query_param(it->second);  
04847     }  
04848     return query;  
04849 }  
04850  
04851 inline void parse_query_text(const char *data, std::size_t size,  
04852                                     Params &params) {  
04853     std::set<std::string> cache;  
04854     split(data, data + size, '&', [&](const char *b, const char *e) {  
04855         std::string kv(b, e);  
04856         if (cache.find(kv) != cache.end()) { return; }  
04857         cache.insert(std::move(kv));  
04858  
04859         std::string key;  
04860         std::string val;  
04861         divide(b, static_cast<std::size_t>(e - b), '=',  
04862             [&](const char *lhs_data, std::size_t lhs_size, const char *rhs_data,  
04863                 std::size_t rhs_size) {  
04864             key.assign(lhs_data, lhs_size);  
04865             val.assign(rhs_data, rhs_size);  
04866         });  
04867  
04868         if (!key.empty()) {  
04869             params.emplace(decode_url(key, true), decode_url(val, true));  
04870         }  
04871     });  
04872 }  
04873  
04874 inline void parse_query_text(const std::string &s, Params &params) {  
04875     parse_query_text(s.data(), s.size(), params);  
04876 }  
04877  
04878 inline bool parse_multipart_boundary(const std::string &content_type,  
04879                                         std::string &boundary) {  
04880     auto boundary_keyword = "boundary=";  
04881     auto pos = content_type.find(boundary_keyword);  
04882     if (pos == std::string::npos) { return false; }  
04883     auto end = content_type.find(';', pos);  
04884     auto beg = pos + strlen(boundary_keyword);  
04885     boundary = trim_double_quotes_copy(content_type.substr(beg, end - beg));  
04886     return !boundary.empty();  
04887 }  
04888  
04889 inline void parse_disposition_params(const std::string &s, Params &params) {  
04890     std::set<std::string> cache;  
04891     split(s.data(), s.data() + s.size(), ';', [&](const char *b, const char *e) {  
04892         std::string kv(b, e);  
04893         if (cache.find(kv) != cache.end()) { return; }  
04894         cache.insert(kv);  
04895  
04896         std::string key;  
04897         std::string val;  
04898         split(b, e, '=', [&](const char *b2, const char *e2) {  
04899             if (key.empty()) {  
04900                 key.assign(b2, e2);  
04901             } else {  
04902                 val.assign(b2, e2);  
04903             }  
04904         });  
04905  
04906         if (!key.empty()) {  
04907             params.emplace(trim_double_quotes_copy((key)),
```

```

04908         trim_double_quotes_copy((val)));
04909     }
04910   });
04911 }
04912
04913 #ifdef CPPHTTPLIB_NO_EXCEPTIONS
04914 inline bool parse_range_header(const std::string &s, Ranges &ranges) {
04915 #else
04916 inline bool parse_range_header(const std::string &s, Ranges &ranges) try {
04917 #endif
04918   auto is_valid = [](<const std::string &str) {
04919     return std::all_of(str.cbegin(), str.cend(),
04920                        [<](unsigned char c) { return std::isdigit(c); });
04921   };
04922
04923   if (s.size() > 7 && s.compare(0, 6, "bytes=") == 0) {
04924     const auto pos = static_cast<size_t>(6);
04925     const auto len = static_cast<size_t>(s.size() - 6);
04926     auto all_valid_ranges = true;
04927     split(&s[pos], &s[pos + len], ',', [&](const char *b, const char *e) {
04928       if (!all_valid_ranges) { return; }
04929
04930       const auto it = std::find(b, e, ',');
04931       if (it == e) {
04932         all_valid_ranges = false;
04933         return;
04934       }
04935
04936       const auto lhs = std::string(b, it);
04937       const auto rhs = std::string(it + 1, e);
04938       if (!is_valid(lhs) || !is_valid(rhs)) {
04939         all_valid_ranges = false;
04940         return;
04941       }
04942
04943       const auto first =
04944         static_cast<ssize_t>(lhs.empty() ? -1 : std::stoll(lhs));
04945       const auto last =
04946         static_cast<ssize_t>(rhs.empty() ? -1 : std::stoll(rhs));
04947       if ((first == -1 && last == -1) ||
04948           (first != -1 && last != -1 && first > last)) {
04949         all_valid_ranges = false;
04950         return;
04951     }
04952
04953     ranges.emplace_back(first, last);
04954   });
04955   return all_valid_ranges && !ranges.empty();
04956 }
04957 return false;
04958 #endif CPPHTTPLIB_NO_EXCEPTIONS
04959 }
04960 #else
04961 } catch (...) { return false; }
04962 #endif
04963
04964 class MultipartFormDataParser {
04965 public:
04966   MultipartFormDataParser() = default;
04967
04968   void set_boundary(std::string &&boundary) {
04969     boundary_ = boundary;
04970     dash_boundary_crlf_ = dash_ + boundary_ + crlf_;
04971     crlf_dash_boundary_ = crlf_ + dash_ + boundary_;
04972   }
04973
04974   bool is_valid() const { return is_valid_; }
04975
04976   bool parse(const char *buf, size_t n, const ContentReceiver &content_callback,
04977             const MultipartContentHeader &header_callback) {
04978
04979     buf_append(buf, n);
04980
04981     while (buf_size() > 0) {
04982       switch (state_) {
04983         case 0: { // Initial boundary
04984           buf_erase(buf_find(dash_boundary_crlf_));
04985           if (dash_boundary_crlf_.size() > buf_size()) { return true; }
04986           if (!buf_start_with(dash_boundary_crlf_)) { return false; }
04987           buf_erase(dash_boundary_crlf_.size());
04988           state_ = 1;
04989           break;
04990         }
04991         case 1: { // New entry
04992           clear_file_info();
04993           state_ = 2;
04994           break;

```

```

04995      }
04996  case 2: { // Headers
04997    auto pos = buf_find(crlf_);
04998    if (pos > CPPHTTPLIB_HEADER_MAX_LENGTH) { return false; }
04999    while (pos < buf_size()) {
05000      // Empty line
05001      if (pos == 0) {
05002        if (!header_callback(file_)) {
05003          is_valid_ = false;
05004          return false;
05005        }
05006        buf_erase(crlf_.size());
05007        state_ = 3;
05008        break;
05009      }
05010
05011      const auto header = buf_head(pos);
05012
05013      if (!parse_header(header.data(), header.data() + header.size(),
05014          [&](const std::string &, const std::string &) {})) {
05015        is_valid_ = false;
05016        return false;
05017      }
05018
05019      constexpr const char header_content_type[] = "Content-Type:";
05020
05021      if (start_with_case_ignore(header, header_content_type)) {
05022        file_.content_type =
05023          trim_copy(header.substr(str_len(header_content_type)));
05024      } else {
05025        thread_local const std::regex re_content_disposition(
05026          R"~((Content-Disposition:\s*form-data;\s*(.*))~",
05027          std::regex_constants::icase);
05028
05029        std::smatch m;
05030        if (std::regex_match(header, m, re_content_disposition)) {
05031          Params params;
05032          parse_disposition_params(m[1], params);
05033
05034          auto it = params.find("name");
05035          if (it != params.end()) {
05036            file_.name = it->second;
05037          } else {
05038            is_valid_ = false;
05039            return false;
05040          }
05041
05042          it = params.find("filename");
05043          if (it != params.end()) { file_.filename = it->second; }
05044
05045          it = params.find("filename*");
05046          if (it != params.end()) {
05047            // Only allow UTF-8 encoding...
05048            thread_local const std::regex re_rfc5987_encoding(
05049              R"~(^UTF-8"(.)$)~", std::regex_constants::icase);
05050
05051            std::smatch m2;
05052            if (std::regex_match(it->second, m2, re_rfc5987_encoding)) {
05053              file_.filename = decode_url(m2[1], false); // override...
05054            } else {
05055              is_valid_ = false;
05056              return false;
05057            }
05058          }
05059        }
05060
05061        buf_erase(pos + crlf_.size());
05062        pos = buf_find(crlf_);
05063      }
05064      if (state_ != 3) { return true; }
05065      break;
05066    }
05067  case 3: { // Body
05068    if (crlf_dash_boundary_.size() > buf_size()) { return true; }
05069    auto pos = buf_find(crlf_dash_boundary_);
05070    if (pos < buf_size()) {
05071      if (!content_callback(buf_data(), pos)) {
05072        is_valid_ = false;
05073        return false;
05074      }
05075      buf_erase(pos + crlf_dash_boundary_.size());
05076      state_ = 4;
05077    } else {
05078      auto len = buf_size() - crlf_dash_boundary_.size();
05079      if (len > 0) {
05080        if (!content_callback(buf_data(), len)) {
05081          is_valid_ = false;
05082        }
05083      }
05084    }
05085  }

```

```

05082         return false;
05083     }
05084     buf_erase(len);
05085   }
05086   return true;
05087 }
05088 break;
05089 }
05090 case 4: { // Boundary
05091   if (crlf_.size() > buf_size()) { return true; }
05092   if (buf_start_with(crlf_)) {
05093     buf_erase(crlf_.size());
05094     state_ = 1;
05095   } else {
05096     if (dash_.size() > buf_size()) { return true; }
05097     if (buf_start_with(dash_)) {
05098       buf_erase(dash_.size());
05099       is_valid_ = true;
05100       buf_erase(buf_size()); // Remove epilogue
05101     } else {
05102       return true;
05103     }
05104   }
05105 break;
05106 }
05107 }
05108 }
05109 return true;
05110 }
05111 }
05112
05113 private:
05114 void clear_file_info() {
05115   file_.name.clear();
05116   file_.filename.clear();
05117   file_.content_type.clear();
05118 }
05119
05120 bool start_with_case_ignore(const std::string &a, const char *b) const {
05121   const auto b_len = strlen(b);
05122   if (a.size() < b_len) { return false; }
05123   for (size_t i = 0; i < b_len; i++) {
05124     if (case_ignore::to_lower(a[i]) != case_ignore::to_lower(b[i])) {
05125       return false;
05126     }
05127   }
05128   return true;
05129 }
05130
05131 const std::string dash_ = "--";
05132 const std::string crlf_ = "\r\n";
05133 std::string boundary_;
05134 std::string dash_boundary_crlf_;
05135 std::string crlf_dash_boundary_;
05136
05137 size_t state_ = 0;
05138 bool is_valid_ = false;
05139 MultipartFormData file_;
05140
05141 // Buffer
05142 bool start_with(const std::string &a, size_t spos, size_t epos,
05143   const std::string &b) const {
05144   if (epos - spos < b.size()) { return false; }
05145   for (size_t i = 0; i < b.size(); i++) {
05146     if (a[i + spos] != b[i]) { return false; }
05147   }
05148   return true;
05149 }
05150
05151 size_t buf_size() const { return buf_epos_ - buf_spos_; }
05152
05153 const char *buf_data() const { return &buf_[buf_spos_]; }
05154
05155 std::string buf_head(size_t l) const { return buf_.substr(buf_spos_, l); }
05156
05157 bool buf_start_with(const std::string &s) const {
05158   return start_with(buf_, buf_spos_, buf_epos_, s);
05159 }
05160
05161 size_t buf_find(const std::string &s) const {
05162   auto c = s.front();
05163
05164   size_t off = buf_spos_;
05165   while (off < buf_epos_) {
05166     auto pos = off;
05167     while (true) {
05168       if (pos == buf_epos_) { return buf_size(); }

```

```

05169     if (buf_[pos] == c) { break; }
05170     pos++;
05171   }
05172 
05173   auto remaining_size = buf_.epos_ - pos;
05174   if (s.size() > remaining_size) { return buf_.size(); }
05175 
05176   if (start_with(buf_, pos, buf_.epos_, s)) { return pos - buf_.spos_; }
05177 
05178   off = pos + 1;
05179 }
05180 
05181   return buf_.size();
05182 }
05183 
05184 void buf_.append(const char *data, size_t n) {
05185   auto remaining_size = buf_.size();
05186   if (remaining_size > 0 && buf_.spos_ > 0) {
05187     for (size_t i = 0; i < remaining_size; i++) {
05188       buf_[i] = buf_[buf_.spos_ + i];
05189     }
05190   }
05191   buf_.spos_ = 0;
05192   buf_.epos_ = remaining_size;
05193 
05194   if (remaining_size + n > buf_.size()) { buf_.resize(remaining_size + n); }
05195 
05196   for (size_t i = 0; i < n; i++) {
05197     buf_[buf_.epos_ + i] = data[i];
05198   }
05199   buf_.epos_ += n;
05200 }
05201 
05202 void buf_.erase(size_t size) { buf_.spos_ += size; }
05203 
05204 std::string buf_;
05205 size_t buf_.spos_ = 0;
05206 size_t buf_.epos_ = 0;
05207 };
05208 
05209 inline std::string random_string(size_t length) {
05210   constexpr const char data[] =
05211     "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
05212 
05213   thread_local auto engine([]() {
05214     // std::random_device might actually be deterministic on some
05215     // platforms, but due to lack of support in the C++ standard library,
05216     // doing better requires either some ugly hacks or breaking portability.
05217     std::random_device seed_gen;
05218     // Request 128 bits of entropy for initialization
05219     std::seed_seq seed_sequence{seed_gen(), seed_gen(), seed_gen(), seed_gen()};
05220     return std::mt19937(seed_sequence);
05221   }());
05222 
05223   std::string result;
05224   for (size_t i = 0; i < length; i++) {
05225     result += data[engine() % (sizeof(data) - 1)];
05226   }
05227   return result;
05228 }
05229 
05230 inline std::string make_multipart_data_boundary() {
05231   return "--cpp-httplib-multipart-data-" + detail::random_string(16);
05232 }
05233 
05234 inline bool is_multipart_boundary_chars_valid(const std::string &boundary) {
05235   auto valid = true;
05236   for (size_t i = 0; i < boundary.size(); i++) {
05237     auto c = boundary[i];
05238     if (!std::isalnum(c) && c != '-' && c != '_') {
05239       valid = false;
05240       break;
05241     }
05242   }
05243   return valid;
05244 }
05245 
05246 template <typename T>
05247 inline std::string
05248 serialize_multipart_formdata_item(const T &item,
05249                                     const std::string &boundary) {
05250   std::string body = "--" + boundary + "\r\n";
05251   body += "Content-Disposition: form-data; name=\"" + item.name + "\"";
05252   if (item.filename.empty()) {
05253     body += "; filename=\"" + item.filename + "\"";
05254   }
05255   body += "\r\n";

```

```
05256 if (!item.content_type.empty()) {
05257     body += "Content-Type: " + item.content_type + "\r\n";
05258 }
05259 body += "\r\n";
05260
05261 return body;
05262 }
05263
05264 inline std::string serialize_multipart_formdata_item_end() { return "\r\n"; }
05265
05266 inline std::string
05267 serialize_multipart_formdata_finish(const std::string &boundary) {
05268     return "--" + boundary + "--\r\n";
05269 }
05270
05271 inline std::string
05272 serialize_multipart_formdata_get_content_type(const std::string &boundary) {
05273     return "multipart/form-data; boundary=" + boundary;
05274 }
05275
05276 inline std::string
05277 serialize_multipart_formdata(const MultipartFormDataItems &items,
05278                                 const std::string &boundary, bool finish = true) {
05279     std::string body;
05280
05281     for (const auto &item : items) {
05282         body += serialize_multipart_formdata_item_begin(item, boundary);
05283         body += item.content + serialize_multipart_formdata_item_end();
05284     }
05285
05286     if (finish) { body += serialize_multipart_formdata_finish(boundary); }
05287
05288     return body;
05289 }
05290
05291 inline bool range_error(Request &req, Response &res) {
05292     if (!req.ranges.empty() && 200 <= res.status && res.status < 300) {
05293         ssize_t content_len = static_cast<ssize_t>(
05294             res.content_length_ ? res.content_length_ : res.body.size());
05295
05296         ssize_t prev_first_pos = -1;
05297         ssize_t prev_last_pos = -1;
05298         size_t overwrapping_count = 0;
05299
05300         // NOTE: The following Range check is based on '14.2. Range' in RFC 9110
05301         // 'HTTP Semantics' to avoid potential denial-of-service attacks.
05302         // https://www.rfc-editor.org/rfc/rfc9110#section-14.2
05303
05304         // Too many ranges
05305         if (req.ranges.size() > CPPHTTPLIB_RANGE_MAX_COUNT) { return true; }
05306
05307         for (auto &r : req.ranges) {
05308             auto &first_pos = r.first;
05309             auto &last_pos = r.second;
05310
05311             if (first_pos == -1 && last_pos == -1) {
05312                 first_pos = 0;
05313                 last_pos = content_len;
05314             }
05315
05316             if (first_pos == -1) {
05317                 first_pos = content_len - last_pos;
05318                 last_pos = content_len - 1;
05319             }
05320
05321             // NOTE: RFC-9110 '14.1.2. Byte Ranges':
05322             // A client can limit the number of bytes requested without knowing the
05323             // size of the selected representation. If the last-pos value is absent,
05324             // or if the value is greater than or equal to the current length of the
05325             // representation data, the byte range is interpreted as the remainder of
05326             // the representation (i.e., the server replaces the value of last-pos
05327             // with a value that is one less than the current length of the selected
05328             // representation).
05329             // https://www.rfc-editor.org/rfc/rfc9110.html#section-14.1.2-6
05330             if (last_pos == -1 || last_pos >= content_len) {
05331                 last_pos = content_len - 1;
05332             }
05333
05334             // Range must be within content length
05335             if (!(0 <= first_pos && first_pos <= last_pos &&
05336                  last_pos <= content_len - 1)) {
05337                 return true;
05338             }
05339
05340             // Ranges must be in ascending order
05341             if (first_pos <= prev_first_pos) { return true; }
05342 }
```

```

05343 // Request must not have more than two overlapping ranges
05344 if (first_pos <= prev_last_pos) {
05345     overwrapping_count++;
05346     if (overwrapping_count > 2) { return true; }
05347 }
05348
05349 prev_first_pos = (std::max)(prev_first_pos, first_pos);
05350 prev_last_pos = (std::max)(prev_last_pos, last_pos);
05351 }
05352 }
05353
05354 return false;
05355 }
05356
05357 inline std::pair<size_t, size_t>
05358 get_range_offset_and_length(Range r, size_t content_length) {
05359     assert(r.first != -1 && r.second != -1);
05360     assert(0 <= r.first && r.first < static_cast<ssize_t>(content_length));
05361     assert(r.first <= r.second &&
05362             r.second < static_cast<ssize_t>(content_length));
05363     (void)(content_length);
05364     return std::make_pair(r.first, static_cast<size_t>(r.second - r.first) + 1);
05365 }
05366
05367 inline std::string make_content_range_header_field(
05368     const std::pair<size_t, size_t> &offset_and_length, size_t content_length) {
05369     auto st = offset_and_length.first;
05370     auto ed = st + offset_and_length.second - 1;
05371
05372     std::string field = "bytes ";
05373     field += std::to_string(st);
05374     field += "-";
05375     field += std::to_string(ed);
05376     field += "/";
05377     field += std::to_string(content_length);
05378     return field;
05379 }
05380
05381 template <typename SToken, typename CToken, typename Content>
05382 bool process_multipart_ranges_data(const Request &req,
05383                                     const std::string &boundary,
05384                                     const std::string &content_type,
05385                                     size_t content_length, SToken stoken,
05386                                     CToken ctoken, Content content) {
05387     for (size_t i = 0; i < req.ranges.size(); i++) {
05388         ctoken("--");
05389         stoken(boundary);
05390         ctoken("\r\n");
05391         if (!content_type.empty()) {
05392             ctoken("Content-Type: ");
05393             stoken(content_type);
05394             ctoken("\r\n");
05395         }
05396         auto offset_and_length =
05397             get_range_offset_and_length(req.ranges[i], content_length);
05398
05399         ctoken("Content-Range: ");
05400         stoken(make_content_range_header_field(offset_and_length, content_length));
05401         ctoken("\r\n");
05402         ctoken("\r\n");
05403
05404         if (!content(offset_and_length.first, offset_and_length.second)) {
05405             return false;
05406         }
05407         ctoken("\r\n");
05408     }
05409 }
05410
05411 ctoken("--");
05412 stoken(boundary);
05413 ctoken("--");
05414
05415 return true;
05416 }
05417
05418 inline void make_multipart_ranges_data(const Request &req, Response &res,
05419                                         const std::string &boundary,
05420                                         const std::string &content_type,
05421                                         size_t content_length,
05422                                         std::string &data) {
05423     process_multipart_ranges_data(
05424         req, boundary, content_type, content_length,
05425         [&](const std::string &token) { data += token; },
05426         [&](const std::string &token) { data += token; },
05427         [&](size_t offset, size_t length) {
05428             assert(offset + length <= content_length);
05429             data += res.body.substr(offset, length);

```

```

05430     return true;
05431   });
05432 }
05433
05434 inline size_t get_multipart_ranges_data_length(const Request &req,
05435                                     const std::string &boundary,
05436                                     const std::string &content_type,
05437                                     size_t content_length) {
05438   size_t data_length = 0;
05439
05440   process_multipart_ranges_data(
05441     req, boundary, content_type, content_length,
05442     [&](const std::string &token) { data_length += token.size(); },
05443     [&](const std::string &token) { data_length += token.size(); },
05444     [&](size_t /*offset*/, size_t length) {
05445       data_length += length;
05446       return true;
05447     });
05448
05449   return data_length;
05450 }
05451
05452 template <typename T>
05453 inline bool
05454 write_multipart_ranges_data(Stream &strm, const Request &req, Response &res,
05455                                     const std::string &boundary,
05456                                     const std::string &content_type,
05457                                     size_t content_length, const T &is_shutting_down) {
05458   return process_multipart_ranges_data(
05459     req, boundary, content_type, content_length,
05460     [&](const std::string &token) { strm.write(token); },
05461     [&](const std::string &token) { strm.write(token); },
05462     [&](size_t offset, size_t length) {
05463       return write_content(strm, res.content_provider_, offset, length,
05464                             is_shutting_down);
05465     });
05466 }
05467
05468 inline bool expect_content(const Request &req) {
05469   if (req.method == "POST" || req.method == "PUT" || req.method == "PATCH" ||
05470       req.method == "DELETE") {
05471     return true;
05472   }
05473   if (req.has_header("Content-Length") &&
05474       req.get_header_value_u64("Content-Length") > 0) {
05475     return true;
05476   }
05477   if (is_chunked_transfer_encoding(req.headers)) { return true; }
05478   return false;
05479 }
05480
05481 inline bool has_crlf(const std::string &s) {
05482   auto p = s.c_str();
05483   while (*p) {
05484     if (*p == '\r' || *p == '\n') { return true; }
05485     p++;
05486   }
05487   return false;
05488 }
05489
05490 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
05491 inline std::string message_digest(const std::string &s, const EVP_MD *algo) {
05492   auto context = std::unique_ptr<EVP_MD_CTX, decltype(&EVP_MD_CTX_free)>(
05493     EVP_MD_CTX_new(), EVP_MD_CTX_free);
05494
05495   unsigned int hash_length = 0;
05496   unsigned char hash[EVP_MAX_MD_SIZE];
05497
05498   EVP_DigestInit_ex(context.get(), algo, nullptr);
05499   EVP_DigestUpdate(context.get(), s.c_str(), s.size());
05500   EVP_DigestFinal_ex(context.get(), hash, &hash_length);
05501
05502   std::stringstream ss;
05503   for (auto i = 0u; i < hash_length; ++i) {
05504     ss << std::hex << std::setw(2) << std::setfill('0')
05505       << static_cast<unsigned int>(hash[i]);
05506   }
05507
05508   return ss.str();
05509 }
05510
05511 inline std::string MD5(const std::string &s) {
05512   return message_digest(s, EVP_md5());
05513 }
05514
05515 inline std::string SHA_256(const std::string &s) {
05516   return message_digest(s, EVP_sha256());

```

```

05517 }
05518
05519 inline std::string SHA_512(const std::string &s) {
05520     return message_digest(s, EVP_sha512());
05521 }
05522
05523 inline std::pair<std::string, std::string> make_digest_authentication_header(
05524     const Request &req, const std::map<std::string, std::string> &auth,
05525     size_t cnonce_count, const std::string &cnonce, const std::string &username,
05526     const std::string &password, bool is_proxy = false) {
05527     std::string nc;
05528     {
05529         std::stringstream ss;
05530         ss << std::setfill('0') << std::setw(8) << std::hex << cnonce_count;
05531         nc = ss.str();
05532     }
05533
05534     std::string qop;
05535     if (auth.find("qop") != auth.end()) {
05536         qop = auth.at("qop");
05537         if (qop.find("auth-int") != std::string::npos) {
05538             qop = "auth-int";
05539         } else if (qop.find("auth") != std::string::npos) {
05540             qop = "auth";
05541         } else {
05542             qop.clear();
05543         }
05544     }
05545
05546     std::string algo = "MD5";
05547     if (auth.find("algorithm") != auth.end()) { algo = auth.at("algorithm"); }
05548
05549     std::string response;
05550     {
05551         auto H = algo == "SHA-256" ? detail::SHA_256
05552             : algo == "SHA-512" ? detail::SHA_512
05553             : detail::MD5;
05554
05555         auto A1 = username + ":" + auth.at("realm") + ":" + password;
05556
05557         auto A2 = req.method + ":" + req.path;
05558         if (qop == "auth-int") { A2 += ":" + H(req.body); }
05559
05560         if (qop.empty()) {
05561             response = H(H(A1) + ":" + auth.at("nonce") + ":" + H(A2));
05562         } else {
05563             response = H(H(A1) + ":" + auth.at("nonce") + ":" + nc + ":" + cnonce +
05564                 ":" + qop + ":" + H(A2));
05565         }
05566     }
05567
05568     auto opaque = (auth.find("opaque") != auth.end()) ? auth.at("opaque") : "";
05569
05570     auto field = "Digest username=\"" + username + "\", realm=\"" +
05571         auth.at("realm") + "\", nonce=\"" + auth.at("nonce") +
05572         "\", uri=\"" + req.path + "\", algorithm=" + algo +
05573         (qop.empty() ? "", response = "\""
05574             : ", qop=" + qop + ", nc=" + nc + ", cnonce=" + "
05575                 cnonce + "\", response=" + "\"")
05576             + response + "\""
05577             + (opaque.empty() ? "" : ", opaque=" + opaque + "\"");
05578
05579     auto key = is_proxy ? "Proxy-Authorization" : "Authorization";
05580     return std::make_pair(key, field);
05581 }
05582
05583 inline bool is_ssl_peer_could_be_closed(SSL *ssl, socket_t sock) {
05584     detail::set_nonblocking(sock, true);
05585     auto se = detail::scope_exit([&]() { detail::set_nonblocking(sock, false); });
05586
05587     char buf[1];
05588     return !SSL_peek(ssl, buf, 1) &&
05589         SSL_get_error(ssl, 0) == SSL_ERROR_ZERO_RETURN;
05590 }
05591
05592 #ifdef _WIN32
05593 // NOTE: This code came up with the following stackoverflow post:
05594 // https://stackoverflow.com/questions/9507184/can-openssl-on-windows-use-the-system-certificate-store
05595 inline bool load_system_certs_on_windows(X509_STORE *store) {
05596     auto hStore = CertOpenSystemStoreW((HCERTPROV_LEGACY)NULL, L"ROOT");
05597     if (!hStore) { return false; }
05598
05599     auto result = false;
05600     PCCERT_CONTEXT pContext = NULL;
05601     while ((pContext = CertEnumCertificatesInStore(hStore, pContext)) !=
05602         nullptr) {
05603         auto encoded_cert =

```

```
05604     static _cast<const unsigned char *>(pContext->pbCertEncoded);
05605
05606     auto x509 = d2i_X509(NULL, &encoded_cert, pContext->cbCertEncoded);
05607     if (x509) {
05608         X509_STORE_add_cert(store, x509);
05609         X509_free(x509);
05610         result = true;
05611     }
05612 }
05613
05614 CertFreeCertificateContext(pContext);
05615 CertCloseStore(hStore, 0);
05616
05617 return result;
05618 }
05619 #elif defined(CPPHTTPLIB_USE_CERTS_FROM_MACOSX_KEYCHAIN) && defined(__APPLE__)
05620 #if TARGET_OS_OSX
05621 template <typename T>
05622 using CFObjectPtr =
05623     std::unique_ptr<typename std::remove_pointer<T>::type, void (*)(CFTypeRef)>;
05624
05625 inline void cf_object_ptr_deleter(CFTypeRef obj) {
05626     if (obj) { CFRelease(obj); }
05627 }
05628
05629 inline bool retrieve_certs_from_keychain(CFObjectPtr<CFArrayRef> &certs) {
05630     CFStringRef keys[] = {kSecClass, kSecMatchLimit, kSecReturnRef};
05631     CFTypeRef values[] = {kSecClassCertificate, kSecMatchLimitAll,
05632                           kCFBooleanTrue};
05633
05634     CFObjectPtr<CFDictionaryRef> query(
05635         CFDictionaryCreate(nullptr, reinterpret_cast<const void **>(keys),
05636                             sizeof(keys) / sizeof(keys[0]),
05637                             &kCFTypeDictionaryKeyCallBacks,
05638                             &kCFTypeDictionaryValueCallBacks),
05639         cf_object_ptr_deleter);
05640
05641     if (!query) { return false; }
05642
05643     CFTypeRef security_items = nullptr;
05644     if (SecItemCopyMatching(query.get(), &security_items) != errSecSuccess ||
05645         CFArrayGetTypeID() != CFGetTypeID(security_items)) {
05646         return false;
05647     }
05648
05649     certs.reset(reinterpret_cast<CFArrayRef>(security_items));
05650     return true;
05651 }
05652
05653 inline bool retrieve_root_certs_from_keychain(CFObjectPtr<CFArrayRef> &certs) {
05654     CFArrayRef root_security_items = nullptr;
05655     if (SecTrustCopyAnchorCertificates(&root_security_items) != errSecSuccess) {
05656         return false;
05657     }
05658
05659     certs.reset(root_security_items);
05660     return true;
05661 }
05662
05663 inline bool add_certs_to_x509_store(CFArrayRef certs, X509_STORE *store) {
05664     auto result = false;
05665     for (auto i = 0; i < CFArrayGetCount(certs); ++i) {
05666         const auto cert = reinterpret_cast<const SecCertificate *>(
05667             CFArrayGetValueAtIndex(certs, i));
05668
05669         if (SecCertificateGetTypeID() != CFGetTypeID(cert)) { continue; }
05670
05671         CFDataRef cert_data = nullptr;
05672         if (SecItemExport(cert, kSecFormatX509Cert, 0, nullptr, &cert_data) !=
05673             errSecSuccess) {
05674             continue;
05675         }
05676
05677         CFObjectPtr<CFDataRef> cert_data_ptr(cert_data, cf_object_ptr_deleter);
05678
05679         auto encoded_cert = static_cast<const unsigned char *>(
05680             CFDataGetBytePtr(cert_data_ptr.get()));
05681
05682         auto x509 =
05683             d2i_X509(NULL, &encoded_cert, CFDataGetLength(cert_data_ptr.get()));
05684
05685         if (x509) {
05686             X509_STORE_add_cert(store, x509);
05687             X509_free(x509);
05688             result = true;
05689         }
05690     }
```

```
05691
05692     return result;
05693 }
05694
05695 inline bool load_system_certs_on_macos(X509_STORE *store) {
05696     auto result = false;
05697     CFOObjectPtr<CFArrayRef> certs(nullptr, cf_object_ptr_deleter);
05698     if (retrieve_certs_from_keychain(certs) && certs) {
05699         result = add_certs_to_x509_store(certs.get(), store);
05700     }
05701
05702     if (retrieve_root_certs_from_keychain(certs) && certs) {
05703         result = add_certs_to_x509_store(certs.get(), store) || result;
05704     }
05705
05706     return result;
05707 }
05708 #endif // TARGET_OS_OSX
05709 #endif // WIN32
05710 #endif // CPPHTTPLIB_OPENSSL_SUPPORT
05711
05712 #ifdef _WIN32
05713 class WSInit {
05714 public:
05715     WSInit() {
05716         WSADATA wsaData;
05717         if (WSAStartup(0x0002, &wsaData) == 0) is_valid_ = true;
05718     }
05719
05720     ~WSInit() {
05721         if (is_valid_) WSACleanup();
05722     }
05723
05724     bool is_valid_ = false;
05725 };
05726
05727 static WSInit wsinit_;
05728 #endif
05729
05730 inline bool parse_www_authenticate(const Response &res,
05731                                     std::map<std::string, std::string> &auth,
05732                                     bool is_proxy) {
05733     auto auth_key = is_proxy ? "Proxy-Authenticate" : "WWW-Authenticate";
05734     if (res.has_header(auth_key)) {
05735         thread_local auto re =
05736             std::regex(R"~((?:(?:\s*)?(.+?)=(?:"(.?)"|([^\n,]*))))~");
05737         auto s = res.get_header_value(auth_key);
05738         auto pos = s.find(' ');
05739         if (pos != std::string::npos) {
05740             auto type = s.substr(0, pos);
05741             if (type == "Basic") {
05742                 return false;
05743             } else if (type == "Digest") {
05744                 s = s.substr(pos + 1);
05745                 auto beg = std::sregex_iterator(s.begin(), s.end(), re);
05746                 for (auto i = beg; i != std::sregex_iterator(); ++i) {
05747                     const auto &m = *i;
05748                     auto key = s.substr(static_cast<size_t>(m.position(1)),
05749                                         static_cast<size_t>(m.length(1)));
05750                     auto val = m.length(2) > 0
05751                         ? s.substr(static_cast<size_t>(m.position(2)),
05752                             static_cast<size_t>(m.length(2)))
05753                         : s.substr(static_cast<size_t>(m.position(3)),
05754                             static_cast<size_t>(m.length(3)));
05755                     auth[key] = val;
05756                 }
05757                 return true;
05758             }
05759         }
05760     }
05761     return false;
05762 }
05763
05764 class ContentProviderAdapter {
05765 public:
05766     explicit ContentProviderAdapter(
05767         ContentProviderWithoutLength &&content_provider)
05768     : content_provider_(content_provider) {}
05769
05770     bool operator()(size_t offset, size_t, DataSink &sink) {
05771         return content_provider_(offset, sink);
05772     }
05773
05774 private:
05775     ContentProviderWithoutLength content_provider_;
05776 };
05777 }
```

```

05778 } // namespace detail
05779
05780 inline std::string hosted_at(const std::string &hostname) {
05781     std::vector<std::string> addrs;
05782     hosted_at(hostname, addrs);
05783     if (addrs.empty()) { return std::string(); }
05784     return addrs[0];
05785 }
05786
05787 inline void hosted_at(const std::string &hostname,
05788                         std::vector<std::string> &addrs) {
05789     struct addrinfo hints;
05790     struct addrinfo *result;
05791
05792     memset(&hints, 0, sizeof(struct addrinfo));
05793     hints.ai_family = AF_UNSPEC;
05794     hints.ai_socktype = SOCK_STREAM;
05795     hints.ai_protocol = 0;
05796
05797     if (getaddrinfo(hostname.c_str(), nullptr, &hints, &result)) {
05798 #if defined __linux__ && !defined __ANDROID__
05799     res_init();
05800 #endif
05801     return;
05802 }
05803 auto se = detail::scope_exit([&] { freeaddrinfo(result); });
05804
05805 for (auto rp = result; rp; rp = rp->ai_next) {
05806     const auto &addr =
05807         *reinterpret_cast<struct sockaddr_storage *>(rp->ai_addr);
05808     std::string ip;
05809     auto dummy = -1;
05810     if (detail::get_ip_and_port(addr, sizeof(struct sockaddr_storage), ip,
05811                                   dummy)) {
05812         addrs.push_back(ip);
05813     }
05814 }
05815 }
05816
05817 inline std::string append_query_params(const std::string &path,
05818                                         const Params &params) {
05819     std::string path_with_query = path;
05820     thread_local const std::regex re("[?]+\\?*");
05821     auto delm = std::regex_match(path, re) ? '&' : '?';
05822     path_with_query += delm + detail::params_to_query_str(params);
05823     return path_with_query;
05824 }
05825
05826 // Header utilities
05827 inline std::pair<std::string, std::string>
05828 make_range_header(const Ranges &ranges) {
05829     std::string field = "bytes=";
05830     auto i = 0;
05831     for (const auto &r : ranges) {
05832         if (i != 0) { field += ", ";}
05833         if (r.first != -1) { field += std::to_string(r.first); }
05834         field += '-';
05835         if (r.second != -1) { field += std::to_string(r.second); }
05836         i++;
05837     }
05838     return std::make_pair("Range", std::move(field));
05839 }
05840
05841 inline std::pair<std::string, std::string>
05842 make_basic_authentication_header(const std::string &username,
05843                                     const std::string &password, bool is_proxy) {
05844     auto field = "Basic " + detail::base64_encode(username + ":" + password);
05845     auto key = is_proxy ? "Proxy-Authorization" : "Authorization";
05846     return std::make_pair(key, std::move(field));
05847 }
05848
05849 inline std::pair<std::string, std::string>
05850 make_bearer_token_authentication_header(const std::string &token,
05851                                             bool is_proxy = false) {
05852     auto field = "Bearer " + token;
05853     auto key = is_proxy ? "Proxy-Authorization" : "Authorization";
05854     return std::make_pair(key, std::move(field));
05855 }
05856
05857 // Request implementation
05858 inline bool Request::has_header(const std::string &key) const {
05859     return detail::has_header(headers, key);
05860 }
05861
05862 inline std::string Request::get_header_value(const std::string &key,
05863                                              const char *def, size_t id) const {
05864     return detail::get_header_value(headers, key, def, id);

```

```
05865 }
05866
05867 inline size_t Request::get_header_value_count(const std::string &key) const {
05868     auto r = headers.equal_range(key);
05869     return static_cast<size_t>(std::distance(r.first, r.second));
05870 }
05871
05872 inline void Request::set_header(const std::string &key,
05873                             const std::string &val) {
05874     if (detail::fields::is_field_name(key) &&
05875         detail::fields::is_field_value(val)) {
05876         headers.emplace(key, val);
05877     }
05878 }
05879
05880 inline bool Request::has_param(const std::string &key) const {
05881     return params.find(key) != params.end();
05882 }
05883
05884 inline std::string Request::get_param_value(const std::string &key,
05885                                         size_t id) const {
05886     auto rng = params.equal_range(key);
05887     auto it = rng.first;
05888     std::advance(it, static_cast<ssize_t>(id));
05889     if (it != rng.second) { return it->second; }
05890     return std::string();
05891 }
05892
05893 inline size_t Request::get_param_value_count(const std::string &key) const {
05894     auto r = params.equal_range(key);
05895     return static_cast<size_t>(std::distance(r.first, r.second));
05896 }
05897
05898 inline bool Request::is_multipart_form_data() const {
05899     const auto &content_type = get_header_value("Content-Type");
05900     return !content_type.rfind("multipart/form-data", 0);
05901 }
05902
05903 inline bool Request::has_file(const std::string &key) const {
05904     return files.find(key) != files.end();
05905 }
05906
05907 inline MultipartFormData Request::get_file_value(const std::string &key) const {
05908     auto it = files.find(key);
05909     if (it != files.end()) { return it->second; }
05910     return MultipartFormData();
05911 }
05912
05913 inline std::vector<MultipartFormData>
05914 Request::get_file_values(const std::string &key) const {
05915     std::vector<MultipartFormData> values;
05916     auto rng = files.equal_range(key);
05917     for (auto it = rng.first; it != rng.second; it++) {
05918         values.push_back(it->second);
05919     }
05920     return values;
05921 }
05922
05923 // Response implementation
05924 inline bool Response::has_header(const std::string &key) const {
05925     return headers.find(key) != headers.end();
05926 }
05927
05928 inline std::string Response::get_header_value(const std::string &key,
05929                                                 const char *def,
05930                                                 size_t id) const {
05931     return detail::get_header_value(headers, key, def, id);
05932 }
05933
05934 inline size_t Response::get_header_value_count(const std::string &key) const {
05935     auto r = headers.equal_range(key);
05936     return static_cast<size_t>(std::distance(r.first, r.second));
05937 }
05938
05939 inline void Response::set_header(const std::string &key,
05940                                     const std::string &val) {
05941     if (detail::fields::is_field_name(key) &&
05942         detail::fields::is_field_value(val)) {
05943         headers.emplace(key, val);
05944     }
05945 }
05946
05947 inline void Response::set_redirect(const std::string &url, int stat) {
05948     if (detail::fields::is_field_value(url)) {
05949         set_header("Location", url);
05950         if (300 <= stat && stat < 400) {
05951             this->status = stat;
05952         }
05953     }
05954 }
```

```

05952     } else {
05953         this->status = StatusCode::Found_302;
05954     }
05955 }
05956 }
05957
05958 inline void Response::set_content(const char *s, size_t n,
05959                     const std::string &content_type) {
05960     body.assign(s, n);
05961
05962     auto rng = headers.equal_range("Content-Type");
05963     headers.erase(rng.first, rng.second);
05964     set_header("Content-Type", content_type);
05965 }
05966
05967 inline void Response::set_content(const std::string &s,
05968                     const std::string &content_type) {
05969     set_content(s.data(), s.size(), content_type);
05970 }
05971
05972 inline void Response::set_content(std::string &s,
05973                     const std::string &content_type) {
05974     body = std::move(s);
05975
05976     auto rng = headers.equal_range("Content-Type");
05977     headers.erase(rng.first, rng.second);
05978     set_header("Content-Type", content_type);
05979 }
05980
05981 inline void Response::set_content_provider(
05982     size_t in_length, const std::string &content_type, ContentProvider provider,
05983     ContentProviderResourceReleaser resource_releaser) {
05984     set_header("Content-Type", content_type);
05985     content_length_ = in_length;
05986     if (in_length > 0) { content_provider_ = std::move(provider); }
05987     content_provider_resource_releaser_ = std::move(resource_releaser);
05988     is_chunked_content_provider_ = false;
05989 }
05990
05991 inline void Response::set_content_provider(
05992     const std::string &content_type, ContentProviderWithoutLength provider,
05993     ContentProviderResourceReleaser resource_releaser) {
05994     set_header("Content-Type", content_type);
05995     content_length_ = 0;
05996     content_provider_ = detail::ContentProviderAdapter(std::move(provider));
05997     content_provider_resource_releaser_ = std::move(resource_releaser);
05998     is_chunked_content_provider_ = false;
05999 }
06000
06001 inline void Response::set_chunked_content_provider(
06002     const std::string &content_type, ContentProviderWithoutLength provider,
06003     ContentProviderResourceReleaser resource_releaser) {
06004     set_header("Content-Type", content_type);
06005     content_length_ = 0;
06006     content_provider_ = detail::ContentProviderAdapter(std::move(provider));
06007     content_provider_resource_releaser_ = std::move(resource_releaser);
06008     is_chunked_content_provider_ = true;
06009 }
06010
06011 inline void Response::set_file_content(const std::string &path,
06012                     const std::string &content_type) {
06013     file_content_path_ = path;
06014     file_content_content_type_ = content_type;
06015 }
06016
06017 inline void Response::set_file_content(const std::string &path) {
06018     file_content_path_ = path;
06019 }
06020
06021 // Result implementation
06022 inline bool Result::has_request_header(const std::string &key) const {
06023     return request_headers_.find(key) != request_headers_.end();
06024 }
06025
06026 inline std::string Result::get_request_header_value(const std::string &key,
06027                     const char *def,
06028                     size_t id) const {
06029     return detail::get_header_value(request_headers_, key, def, id);
06030 }
06031
06032 inline size_t
06033 Result::get_request_header_value_count(const std::string &key) const {
06034     auto r = request_headers_.equal_range(key);
06035     return static_cast<size_t>(std::distance(r.first, r.second));
06036 }
06037
06038 // Stream implementation

```

```

06039 inline ssize_t Stream::write(const char *ptr) {
06040     return write(ptr, strlen(ptr));
06041 }
06042
06043 inline ssize_t Stream::write(const std::string &s) {
06044     return write(s.data(), s.size());
06045 }
06046
06047 namespace detail {
06048
06049 inline void calc_actual_timeout(time_t max_timeout_msec, time_t duration_msec,
06050             time_t timeout_sec, time_t timeout_usec,
06051             time_t &actual_timeout_sec,
06052             time_t &actual_timeout_usec) {
06053     auto timeout_msec = (timeout_sec * 1000) + (timeout_usec / 1000);
06054
06055     auto actual_timeout_msec =
06056         (std::min)(max_timeout_msec - duration_msec, timeout_msec);
06057
06058     if (actual_timeout_msec < 0) { actual_timeout_msec = 0; }
06059
06060     actual_timeout_sec = actual_timeout_msec / 1000;
06061     actual_timeout_usec = (actual_timeout_msec % 1000) * 1000;
06062 }
06063
06064 // Socket stream implementation
06065 inline SocketStream::SocketStream(
06066     socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec,
06067     time_t write_timeout_sec, time_t write_timeout_usec,
06068     time_t max_timeout_msec,
06069     std::chrono::time_point<std::chrono::steady_clock> start_time)
06070 : sock_(sock), read_timeout_sec_(read_timeout_sec),
06071   read_timeout_usec_(read_timeout_usec),
06072   write_timeout_sec_(write_timeout_sec),
06073   write_timeout_usec_(write_timeout_usec),
06074   max_timeout_msec_(max_timeout_msec), start_time_(start_time),
06075   read_buff_(read_buff_size_, 0) {}
06076
06077 inline SocketStream::~SocketStream() = default;
06078
06079 inline bool SocketStream::is_readable() const {
06080     return read_buff_off_ < read_buff_content_size_;
06081 }
06082
06083 inline bool SocketStream::wait_readable() const {
06084     if (max_timeout_msec_ <= 0) {
06085         return select_read(sock_, read_timeout_sec_, read_timeout_usec_) > 0;
06086     }
06087
06088     time_t read_timeout_sec;
06089     time_t read_timeout_usec;
06090     calc_actual_timeout(max_timeout_msec_, duration(), read_timeout_sec_,
06091                         read_timeout_usec_, read_timeout_sec, read_timeout_usec);
06092
06093     return select_read(sock_, read_timeout_sec, read_timeout_usec) > 0;
06094 }
06095
06096 inline bool SocketStream::wait_writable() const {
06097     return select_write(sock_, write_timeout_sec_, write_timeout_usec_) > 0 &&
06098         is_socket_alive(sock_);
06099 }
06100
06101 inline ssize_t SocketStream::read(char *ptr, size_t size) {
06102 #ifdef _WIN32
06103     size =
06104         (std::min)(size, static_cast<size_t>((std::numeric_limits<int>::max)()));
06105 #else
06106     size = (std::min)(size,
06107                     static_cast<size_t>((std::numeric_limits<ssize_t>::max)()));
06108 #endif
06109
06110     if (read_buff_off_ < read_buff_content_size_) {
06111         auto remaining_size = read_buff_content_size_ - read_buff_off_;
06112         if (size <= remaining_size) {
06113             memcpy(ptr, read_buff_.data() + read_buff_off_, size);
06114             read_buff_off_ += size;
06115             return static_cast<ssize_t>(size);
06116         } else {
06117             memcpy(ptr, read_buff_.data() + read_buff_off_, remaining_size);
06118             read_buff_off_ += remaining_size;
06119             return static_cast<ssize_t>(remaining_size);
06120         }
06121     }
06122
06123     if (!wait_readable()) { return -1; }
06124
06125     read_buff_off_ = 0;

```

```
06126     read_buff_content_size_ = 0;
06127
06128     if (size < read_buff_size_) {
06129         auto n = read_socket(sock_, read_buff_.data(), read_buff_size_,
06130                               CPPHTTPLIB_RECV_FLAGS);
06131         if (n <= 0) {
06132             return n;
06133         } else if (n <= static_cast<ssize_t>(size)) {
06134             memcpy(ptr, read_buff_.data(), static_cast<size_t>(n));
06135             return n;
06136         } else {
06137             memcpy(ptr, read_buff_.data(), size);
06138             read_buff_off_ = size;
06139             read_buff_content_size_ = static_cast<size_t>(n);
06140             return static_cast<ssize_t>(size);
06141         }
06142     } else {
06143         return read_socket(sock_, ptr, size, CPPHTTPLIB_RECV_FLAGS);
06144     }
06145 }
06146
06147 inline ssize_t SocketStream::write(const char *ptr, size_t size) {
06148     if (!wait_writable()) { return -1; }
06149
06150 #if defined(_WIN32) && !defined(_WIN64)
06151     size =
06152         (std::min)(size, static_cast<size_t>((std::numeric_limits<int>::max())));
06153 #endif
06154
06155     return send_socket(sock_, ptr, size, CPPHTTPLIB_SEND_FLAGS);
06156 }
06157
06158 inline void SocketStream::get_remote_ip_and_port(std::string &ip,
06159                                                 int &port) const {
06160     return detail::get_remote_ip_and_port(sock_, ip, port);
06161 }
06162
06163 inline void SocketStream::get_local_ip_and_port(std::string &ip,
06164                                                 int &port) const {
06165     return detail::get_local_ip_and_port(sock_, ip, port);
06166 }
06167
06168 inline socket_t SocketStream::socket() const { return sock_; }
06169
06170 inline time_t SocketStream::duration() const {
06171     return std::chrono::duration_cast<std::chrono::milliseconds>(
06172         std::chrono::steady_clock::now() - start_time_)
06173         .count();
06174 }
06175
06176 // Buffer stream implementation
06177 inline bool BufferStream::is_readable() const { return true; }
06178
06179 inline bool BufferStream::wait_readable() const { return true; }
06180
06181 inline bool BufferStream::wait_writable() const { return true; }
06182
06183 inline ssize_t BufferStream::read(char *ptr, size_t size) {
06184 #if defined(_MSC_VER) && _MSC_VER < 1910
06185     auto len_read = buffer._Copy_s(ptr, size, size, position);
06186 #else
06187     auto len_read = buffer.copy(ptr, size, position);
06188 #endif
06189     position += static_cast<size_t>(len_read);
06190     return static_cast<ssize_t>(len_read);
06191 }
06192
06193 inline ssize_t BufferStream::write(const char *ptr, size_t size) {
06194     buffer.append(ptr, size);
06195     return static_cast<ssize_t>(size);
06196 }
06197
06198 inline void BufferStream::get_remote_ip_and_port(std::string & /*ip*/,
06199                                                 int & /*port*/) const {}
06200
06201 inline void BufferStream::get_local_ip_and_port(std::string & /*ip*/,
06202                                                 int & /*port*/) const {}
06203
06204 inline socket_t BufferStream::socket() const { return 0; }
06205
06206 inline time_t BufferStream::duration() const { return 0; }
06207
06208 inline const std::string &BufferStream::get_buffer() const { return buffer; }
06209
06210 inline PathParamsMatcher::PathParamsMatcher(const std::string &pattern) {
06211     constexpr const char marker[] = "/";
06212 }
```

```

06213 // One past the last ending position of a path param substring
06214 std::size_t last_param_end = 0;
06215
06216 #ifndef CPPHTTPLIB_NO_EXCEPTIONS
06217 // Needed to ensure that parameter names are unique during matcher
06218 // construction
06219 // If exceptions are disabled, only last duplicate path
06220 // parameter will be set
06221 std::unordered_set<std::string> param_name_set;
06222 #endif
06223
06224 while (true) {
06225     const auto marker_pos = pattern.find(
06226         marker, last_param_end == 0 ? last_param_end : last_param_end - 1);
06227     if (marker_pos == std::string::npos) { break; }
06228
06229     static_fragments_.push_back(
06230         pattern.substr(last_param_end, marker_pos - last_param_end + 1));
06231
06232     const auto param_name_start = marker_pos + str_len(marker);
06233
06234     auto sep_pos = pattern.find(separator, param_name_start);
06235     if (sep_pos == std::string::npos) { sep_pos = pattern.length(); }
06236
06237     auto param_name =
06238         pattern.substr(param_name_start, sep_pos - param_name_start);
06239
06240 #ifndef CPPHTTPLIB_NO_EXCEPTIONS
06241     if (param_name_set.find(param_name) != param_name_set.end()) {
06242         std::string msg = "Encountered path parameter '" + param_name +
06243             "' multiple times in route pattern '" + pattern + "'";
06244         throw std::invalid_argument(msg);
06245     }
06246 #endif
06247
06248     param_names_.push_back(std::move(param_name));
06249
06250     last_param_end = sep_pos + 1;
06251 }
06252
06253 if (last_param_end < pattern.length()) {
06254     static_fragments_.push_back(pattern.substr(last_param_end));
06255 }
06256 }
06257
06258 inline bool PathParamsMatcher::match(Request &request) const {
06259     request.matches = std::smatch();
06260     request.path_params.clear();
06261     request.path_params.reserve(param_names_.size());
06262
06263     // One past the position at which the path matched the pattern last time
06264     std::size_t starting_pos = 0;
06265     for (size_t i = 0; i < static_fragments_.size(); ++i) {
06266         const auto &fragment = static_fragments_[i];
06267
06268         if (starting_pos + fragment.length() > request.path.length()) {
06269             return false;
06270         }
06271
06272         // Avoid unnecessary allocation by using strncmp instead of substr +
06273         // comparison
06274         if (std::strncmp(request.path.c_str() + starting_pos, fragment.c_str(),
06275                         fragment.length()) != 0) {
06276             return false;
06277         }
06278
06279         starting_pos += fragment.length();
06280
06281         // Should only happen when we have a static fragment after a param
06282         // Example: '/users/:id/subscriptions'
06283         // The 'subscriptions' fragment here does not have a corresponding param
06284         if (i >= param_names_.size()) { continue; }
06285
06286         auto sep_pos = request.path.find(separator, starting_pos);
06287         if (sep_pos == std::string::npos) { sep_pos = request.path.length(); }
06288
06289         const auto &param_name = param_names_[i];
06290
06291         request.path_params.emplace(
06292             param_name, request.path.substr(starting_pos, sep_pos - starting_pos));
06293
06294         // Mark everything up to '/' as matched
06295         starting_pos = sep_pos + 1;
06296     }
06297     // Returns false if the path is longer than the pattern
06298     return starting_pos >= request.path.length();
06299 }
```

```
06300
06301 inline bool RegexMatcher::match(Request &request) const {
06302   request.path_params.clear();
06303   return std::regex_match(request.path, request.matches, regex_);
06304 }
06305
06306 } // namespace detail
06307
06308 // HTTP server implementation
06309 inline Server::Server()
06310   : new_task_queue(
06311     [] { return new ThreadPool(CPPHTTPPLIB_THREAD_POOL_COUNT); }) {
06312 #ifndef WIN32
06313   signal(SIGPIPE, SIG_IGN);
06314 #endif
06315 }
06316
06317 inline Server::~Server() = default;
06318
06319 inline std::unique_ptr<detail::MatcherBase>
06320 Server::make_matcher(const std::string &pattern) {
06321   if (pattern.find("/") != std::string::npos) {
06322     return detail::make_unique<detail::PathParamsMatcher>(pattern);
06323   } else {
06324     return detail::make_unique<detail::RegexMatcher>(pattern);
06325   }
06326 }
06327
06328 inline Server &Server::Get(const std::string &pattern, Handler handler) {
06329   get_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06330   return *this;
06331 }
06332
06333 inline Server &Server::Post(const std::string &pattern, Handler handler) {
06334   post_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06335   return *this;
06336 }
06337
06338 inline Server &Server::Post(const std::string &pattern,
06339                               HandlerWithContentReader handler) {
06340   post_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06341                                                 std::move(handler));
06342   return *this;
06343 }
06344
06345 inline Server &Server::Put(const std::string &pattern, Handler handler) {
06346   put_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06347   return *this;
06348 }
06349
06350 inline Server &Server::Put(const std::string &pattern,
06351                               HandlerWithContentReader handler) {
06352   put_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06353                                                 std::move(handler));
06354   return *this;
06355 }
06356
06357 inline Server &Server::Patch(const std::string &pattern, Handler handler) {
06358   patch_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06359   return *this;
06360 }
06361
06362 inline Server &Server::Patch(const std::string &pattern,
06363                               HandlerWithContentReader handler) {
06364   patch_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06365                                                 std::move(handler));
06366   return *this;
06367 }
06368
06369 inline Server &Server::Delete(const std::string &pattern, Handler handler) {
06370   delete_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06371   return *this;
06372 }
06373
06374 inline Server &Server::Delete(const std::string &pattern,
06375                               HandlerWithContentReader handler) {
06376   delete_handlers_for_content_reader_.emplace_back(make_matcher(pattern),
06377                                                 std::move(handler));
06378   return *this;
06379 }
06380
06381 inline Server &Server::Options(const std::string &pattern, Handler handler) {
06382   options_handlers_.emplace_back(make_matcher(pattern), std::move(handler));
06383   return *this;
06384 }
06385
06386 inline bool Server::set_base_dir(const std::string &dir,
```

```
06387             const std::string &mount_point) {
06388     return set_mount_point(mount_point, dir);
06389 }
06390
06391 inline bool Server::set_mount_point(const std::string &mount_point,
06392                                     const std::string &dir, Headers headers) {
06393     detail::FileStat stat(dir);
06394     if (stat.is_dir()) {
06395         std::string mnt = !mount_point.empty() ? mount_point : "/";
06396         if (!mnt.empty() && mnt[0] == '/') {
06397             base_dirs_.push_back({mnt, dir, std::move(headers)});
06398             return true;
06399         }
06400     }
06401     return false;
06402 }
06403
06404 inline bool Server::remove_mount_point(const std::string &mount_point) {
06405     for (auto it = base_dirs_.begin(); it != base_dirs_.end(); ++it) {
06406         if (*it->mount_point == mount_point) {
06407             base_dirs_.erase(it);
06408             return true;
06409         }
06410     }
06411     return false;
06412 }
06413
06414 inline Server &
06415 Server::set_file_extension_and_mimetype_mapping(const std::string &ext,
06416                                                 const std::string &mime) {
06417     file_extension_and_mimetype_map_[ext] = mime;
06418     return *this;
06419 }
06420
06421 inline Server &Server::set_default_file_mimetype(const std::string &mime) {
06422     default_file_mimetype_ = mime;
06423     return *this;
06424 }
06425
06426 inline Server &Server::set_file_request_handler(Handler handler) {
06427     file_request_handler_ = std::move(handler);
06428     return *this;
06429 }
06430
06431 inline Server &Server::set_error_handler_core(HandlerWithResponse handler,
06432                                                 std::true_type) {
06433     error_handler_ = std::move(handler);
06434     return *this;
06435 }
06436
06437 inline Server &Server::set_error_handler_core(Handler handler,
06438                                                 std::false_type) {
06439     error_handler_ = [handler](const Request &req, Response &res) {
06440         handler(req, res);
06441         return HandlerResponse::Handled;
06442     };
06443     return *this;
06444 }
06445
06446 inline Server &Server::set_exception_handler(ExceptionHandler handler) {
06447     exception_handler_ = std::move(handler);
06448     return *this;
06449 }
06450
06451 inline Server &Server::set_pre_routing_handler(HandlerWithResponse handler) {
06452     pre_routing_handler_ = std::move(handler);
06453     return *this;
06454 }
06455
06456 inline Server &Server::set_post_routing_handler(Handler handler) {
06457     post_routing_handler_ = std::move(handler);
06458     return *this;
06459 }
06460
06461 inline Server &Server::set_logger(Logger logger) {
06462     logger_ = std::move(logger);
06463     return *this;
06464 }
06465
06466 inline Server &
06467 Server::set_expect_100_continue_handler(Expect100ContinueHandler handler) {
06468     expect_100_continue_handler_ = std::move(handler);
06469     return *this;
06470 }
06471
06472 inline Server &Server::set_address_family(int family) {
06473     address_family_ = family;
```

```
06474     return *this;
06475 }
06476
06477 inline Server &Server::set_tcp_nodelay(bool on) {
06478     tcp_nodelay_ = on;
06479     return *this;
06480 }
06481
06482 inline Server &Server::set_ipv6_v6only(bool on) {
06483     ipv6_v6only_ = on;
06484     return *this;
06485 }
06486
06487 inline Server &Server::set_socket_options(SocketOptions socket_options) {
06488     socket_options_ = std::move(socket_options);
06489     return *this;
06490 }
06491
06492 inline Server &Server::set_default_headers(Headers headers) {
06493     default_headers_ = std::move(headers);
06494     return *this;
06495 }
06496
06497 inline Server &Server::set_header_writer(
06498     std::function<ssize_t(Stream &, Headers &) const > writer) {
06499     header_writer_ = writer;
06500     return *this;
06501 }
06502
06503 inline Server &Server::set_keep_alive_max_count(size_t count) {
06504     keep_alive_max_count_ = count;
06505     return *this;
06506 }
06507
06508 inline Server &Server::set_keep_alive_timeout(time_t sec) {
06509     keep_alive_timeout_sec_ = sec;
06510     return *this;
06511 }
06512
06513 inline Server &Server::set_read_timeout(time_t sec, time_t usec) {
06514     read_timeout_sec_ = sec;
06515     read_timeout_usec_ = usec;
06516     return *this;
06517 }
06518
06519 inline Server &Server::set_write_timeout(time_t sec, time_t usec) {
06520     write_timeout_sec_ = sec;
06521     write_timeout_usec_ = usec;
06522     return *this;
06523 }
06524
06525 inline Server &Server::set_idle_interval(time_t sec, time_t usec) {
06526     idle_interval_sec_ = sec;
06527     idle_interval_usec_ = usec;
06528     return *this;
06529 }
06530
06531 inline Server &Server::set_payload_max_length(size_t length) {
06532     payload_max_length_ = length;
06533     return *this;
06534 }
06535
06536 inline bool Server::bind_to_port(const std::string &host, int port,
06537                                     int socket_flags) {
06538     auto ret = bind_internal(host, port, socket_flags);
06539     if (ret == -1) { is_decommissioned_ = true; }
06540     return ret >= 0;
06541 }
06542 inline int Server::bind_to_any_port(const std::string &host, int socket_flags) {
06543     auto ret = bind_internal(host, 0, socket_flags);
06544     if (ret == -1) { is_decommissioned_ = true; }
06545     return ret;
06546 }
06547
06548 inline bool Server::listen_after_bind() { return listen_internal(); }
06549
06550 inline bool Server::listen(const std::string &host, int port,
06551                             int socket_flags) {
06552     return bind_to_port(host, port, socket_flags) && listen_internal();
06553 }
06554
06555 inline bool Server::is_running() const { return is_running_; }
06556
06557 inline void Server::wait_until_ready() const {
06558     while (!is_running_ && !is_decommissioned) {
06559         std::this_thread::sleep_for(std::chrono::milliseconds{1});
06560     }
}
```

```
06561 }
06562
06563 inline void Server::stop() {
06564     if (is_running_) {
06565         assert(svr_sock_ != INVALID_SOCKET);
06566         std::atomic<socket_t> sock(svr_sock_.exchange(INVALID_SOCKET));
06567         detail::shutdown_socket(sock);
06568         detail::close_socket(sock);
06569     }
06570     is_decommissioned = false;
06571 }
06572
06573 inline void Server::decommission() { is_decommissioned = true; }
06574
06575 inline bool Server::parse_request_line(const char *s, Request &req) const {
06576     auto len = strlen(s);
06577     if (len < 2 || s[len - 2] != '\r' || s[len - 1] != '\n') { return false; }
06578     len -= 2;
06579
06580     {
06581         size_t count = 0;
06582
06583         detail::split(s, s + len, ' ', [&](const char *b, const char *e) {
06584             switch (count) {
06585                 case 0: req.method = std::string(b, e); break;
06586                 case 1: req.target = std::string(b, e); break;
06587                 case 2: req.version = std::string(b, e); break;
06588                 default: break;
06589             }
06590             count++;
06591         });
06592
06593     if (count != 3) { return false; }
06594 }
06595
06596 thread_local const std::set<std::string> methods{
06597     "GET", "HEAD", "POST", "PUT", "DELETE",
06598     "CONNECT", "OPTIONS", "TRACE", "PATCH", "PRI";
06599
06600     if (methods.find(req.method) == methods.end()) { return false; }
06601
06602     if (req.version != "HTTP/1.1" && req.version != "HTTP/1.0") { return false; }
06603
06604     {
06605         // Skip URL fragment
06606         for (size_t i = 0; i < req.target.size(); i++) {
06607             if (req.target[i] == '#') {
06608                 req.target.erase(i);
06609                 break;
06610             }
06611         }
06612
06613         detail::divide(req.target, '?',
06614             [&](const char *lhs_data, std::size_t lhs_size,
06615                 const char *rhs_data, std::size_t rhs_size) {
06616                 req.path = detail::decode_url(
06617                     std::string(lhs_data, lhs_size), false);
06618                 detail::parse_query_text(rhs_data, rhs_size, req.params);
06619             });
06620     }
06621
06622     return true;
06623 }
06624
06625 inline bool Server::write_response(Stream &strm, bool close_connection,
06626                                     Request &req, Response &res) {
06627     // NOTE: `req.ranges` should be empty, otherwise it will be applied
06628     // incorrectly to the error content.
06629     req.ranges.clear();
06630     return write_response_core(strm, close_connection, req, res, false);
06631 }
06632
06633 inline bool Server::write_response_with_content(Stream &strm,
06634                                                 bool close_connection,
06635                                                 const Request &req,
06636                                                 Response &res) {
06637     return write_response_core(strm, close_connection, req, res, true);
06638 }
06639
06640 inline bool Server::write_response_core(Stream &strm, bool close_connection,
06641                                         const Request &req, Response &res,
06642                                         bool need_apply_ranges) {
06643     assert(res.status != -1);
06644
06645     if (400 <= res.status && error_handler_ &&
06646         error_handler_(req, res) == HandlerResponse::Handled) {
06647         need_apply_ranges = true;
06648 }
```

```

06648 }
06649 std::string content_type;
06650 std::string boundary;
06651 if (need_apply_ranges) { apply_ranges(req, res, content_type, boundary); }
06653
06654 // Prepare additional headers
06655 if (close_connection || req.get_header_value("Connection") == "close") {
06656 res.set_header("Connection", "close");
06657 } else {
06658 std::string s = "timeout=";
06659 s += std::to_string(keep_alive_timeout_sec_);
06660 s += ", max=";
06661 s += std::to_string(keep_alive_max_count_);
06662 res.set_header("Keep-Alive", s);
06663 }
06664
06665 if (!(res.body.empty() || res.content_length_ > 0 || res.content_provider_) &&
06666 !res.has_header("Content-Type")) {
06667 res.set_header("Content-Type", "text/plain");
06668 }
06669
06670 if (res.body.empty() && !res.content_length_ && !res.content_provider_ &&
06671 !res.has_header("Content-Length")) {
06672 res.set_header("Content-Length", "0");
06673 }
06674
06675 if (req.method == "HEAD" && !res.has_header("Accept-Ranges")) {
06676 res.set_header("Accept-Ranges", "bytes");
06677 }
06678
06679 if (post_routing_handler_) { post_routing_handler_(req, res); }
06680
06681 // Response line and headers
06682 {
06683 detail::BufferStream bstrm;
06684 if (!detail::write_response_line(bstrm, res.status)) { return false; }
06685 if (!header_writer_(bstrm, res.headers)) { return false; }
06686
06687 // Flush buffer
06688 auto &data = bstrm.get_buffer();
06689 detail::write_data(strm, data.data(), data.size());
06690 }
06691
06692 // Body
06693 auto ret = true;
06694 if (req.method != "HEAD") {
06695 if (!res.body.empty()) {
06696 if (!detail::write_data(strm, res.body.data(), res.body.size())) {
06697 ret = false;
06698 }
06699 } else if (res.content_provider_) {
06700 if (write_content_with_provider(strm, req, res, boundary, content_type)) {
06701 res.content_provider_success_ = true;
06702 } else {
06703 ret = false;
06704 }
06705 }
06706 }
06707
06708 // Log
06709 if (logger_) { logger_(req, res); }
06710
06711 return ret;
06712 }
06713
06714 inline bool
06715 Server::write_content_with_provider(Stream &strm, const Request &req,
06716 const Response &res, const std::string &boundary,
06717 const std::string &content_type) {
06718 auto is_shutting_down = [this]() {
06719 return this->svr_sock_ == INVALID_SOCKET;
06720 };
06721
06722 if (res.content_length_ > 0) {
06723 if (req.ranges.empty()) {
06724 return detail::write_content(strm, res.content_provider_, 0,
06725 res.content_length_, is_shutting_down);
06726 } else if (req.ranges.size() == 1) {
06727 auto offset_and_length = detail::get_range_offset_and_length(
06728 req.ranges[0], res.content_length_);
06729
06730 return detail::write_content(strm, res.content_provider_,
06731 offset_and_length.first,
06732 offset_and_length.second, is_shutting_down);
06733 } else {
06734 return detail::write_multipart_ranges_data(

```

```

06735     strm, req, res, boundary, content_type, res.content_length_,
06736     is_shutting_down);
06737 }
06738 } else {
06739     if (res.is_chunked_content_provider_) {
06740         auto type = detail::encoding_type(req, res);
06741
06742         std::unique_ptr<detail::compressor> compressor;
06743         if (type == detail::EncodingType::Gzip) {
06744 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
06745             compressor = detail::make_unique<detail::gzip_compressor>();
06746 #endif
06747         } else if (type == detail::EncodingType::Brotli) {
06748 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
06749             compressor = detail::make_unique<detail::brotli_compressor>();
06750 #endif
06751         } else if (type == detail::EncodingType::Zstd) {
06752 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
06753             compressor = detail::make_unique<detail::zstd_compressor>();
06754 #endif
06755         } else {
06756             compressor = detail::make_unique<detail::nocompressor>();
06757         }
06758         assert(compressor != nullptr);
06759
06760         return detail::write_content_chunked(strm, res.content_provider_,
06761                                             is_shutting_down, *compressor);
06762     } else {
06763         return detail::write_content_without_length(strm, res.content_provider_,
06764                                             is_shutting_down);
06765     }
06766 }
06767 }
06768
06769 inline bool Server::read_content(Stream &strm, Request &req, Response &res) {
06770     MultipartFormDataMap::iterator cur;
06771     auto file_count = 0;
06772     if (read_content_core(
06773         strm, req, res,
06774         // Regular
06775         [&](const char *buf, size_t n) {
06776             if (req.body.size() + n > req.body.max_size()) { return false; }
06777             req.body.append(buf, n);
06778             return true;
06779         },
06780         // Multipart
06781         [&](const MultipartFormData &file) {
06782             if (file_count++ == CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX_COUNT) {
06783                 return false;
06784             }
06785             cur = req.files.emplace(file.name, file);
06786             return true;
06787         },
06788         [&](const char *buf, size_t n) {
06789             auto &content = cur->second.content;
06790             if (content.size() + n > content.max_size()) { return false; }
06791             content.append(buf, n);
06792             return true;
06793         })
06794     const auto &content_type = req.get_header_value("Content-Type");
06795     if (!content_type.find("application/x-www-form-urlencoded")) {
06796         if (req.body.size() > CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MAX_LENGTH) {
06797             res.status = StatusCode::PayloadTooLarge_413; // NOTE: should be 414?
06798             return false;
06799         }
06800         detail::parse_query_text(req.body, req.params);
06801     }
06802     return true;
06803 }
06804 return false;
06805 }
06806
06807 inline bool Server::read_content_with_content_receiver(
06808     Stream &strm, Request &req, Response &res, ContentReceiver receiver,
06809     MultipartContentHeader multipart_header,
06810     ContentReceiver multipart_receiver) {
06811     return read_content_core(strm, req, res, std::move(receiver),
06812                             std::move(multipart_header),
06813                             std::move(multipart_receiver));
06814 }
06815
06816 inline bool
06817 Server::read_content_core(Stream &strm, Request &req, Response &res,
06818                             ContentReceiver receiver,
06819                             MultipartContentHeader multipart_header,
06820                             ContentReceiver multipart_receiver) const {
06821     detail::MultipartFormDataParser multipart_form_data_parser;

```

```

06822 ContentReceiverWithProgress out;
06823 if (req.is_multipart_form_data()) {
06824     const auto &content_type = req.get_header_value("Content-Type");
06825     std::string boundary;
06826     if (!detail::parse_multipart_boundary(content_type, boundary)) {
06827         res.status = StatusCode::BadRequest_400;
06828         return false;
06829     }
06830 }
06831 multipart_form_data_parser.set_boundary(std::move(boundary));
06832 out = [&](const char *buf, size_t n, uint64_t /*off*/, uint64_t /*len*/) {
06833     /* For debug
06834     size_t pos = 0;
06835     while (pos < n) {
06836         auto read_size = (std::min)(size_t)(1, n - pos);
06837         auto ret = multipart_form_data_parser.parse(
06838             buf + pos, read_size, multipart_receiver, multipart_header);
06839         if (!ret) { return false; }
06840         pos += read_size;
06841     }
06842     return true;
06843 */
06844     return multipart_form_data_parser.parse(buf, n, multipart_receiver,
06845                                                 multipart_header);
06846 };
06847 } else {
06848     out = [receiver](const char *buf, size_t n, uint64_t /*off*/,
06849                      uint64_t /*len*/) { return receiver(buf, n); };
06850 }
06851
06852 if (req.method == "DELETE" && !req.has_header("Content-Length")) {
06853     return true;
06854 }
06855
06856 if (!detail::read_content(strm, req, payload_max_length_, res.status, nullptr,
06857                             out, true)) {
06858     return false;
06859 }
06860
06861 if (req.is_multipart_form_data()) {
06862     if (!multipart_form_data_parser.is_valid()) {
06863         res.status = StatusCode::BadRequest_400;
06864         return false;
06865     }
06866 }
06867
06868 return true;
06870 }
06871
06872 inline bool Server::handle_file_request(const Request &req, Response &res,
06873                                         bool head) {
06874     for (const auto &entry : base_dirs_) {
06875         // Prefix match
06876         if (!req.path.compare(0, entry.mount_point.size(), entry.mount_point)) {
06877             std::string sub_path = "/" + req.path.substr(entry.mount_point.size());
06878             if (detail::is_valid_path(sub_path)) {
06879                 auto path = entry.base_dir + sub_path;
06880                 if (path.back() == '/') { path += "index.html"; }
06881
06882                 detail::FileStat stat(path);
06883
06884                 if (stat.is_dir()) {
06885                     res.set_redirect(sub_path + "/", StatusCode::MovedPermanently_301);
06886                     return true;
06887                 }
06888
06889                 if (stat.is_file()) {
06890                     for (const auto &kv : entry.headers) {
06891                         res.set_header(kv.first, kv.second);
06892                     }
06893
06894                     auto mm = std::make_shared<detail::mmap>(path.c_str());
06895                     if (!mm->is_open()) { return false; }
06896
06897                     res.set_content_provider(
06898                         mm->size(),
06899                         detail::find_content_type(path, file_extension_and_mimetype_map_,
06900                                         default_file_mimetype),
06901                         [mm](size_t offset, size_t length, DataSink &sink) -> bool {
06902                             sink.write(mm->data() + offset, length);
06903                             return true;
06904                         });
06905
06906                     if (!head && file_request_handler_) {
06907                         file_request_handler_(req, res);
06908                     }
06909     }
06910 }
06911
06912
06913
06914
06915
06916
06917
06918
06919
06920
06921
06922
06923
06924
06925
06926
06927
06928
06929
06930
06931
06932
06933
06934
06935
06936
06937
06938
06939
06940
06941
06942
06943
06944
06945
06946
06947
06948
06949
06950
06951
06952
06953
06954
06955
06956
06957
06958
06959
06960
06961
06962
06963
06964
06965
06966
06967
06968
06969
06970
06971
06972
06973
06974
06975
06976
06977
06978
06979
06980
06981
06982
06983
06984
06985
06986
06987
06988
06989
06990
06991
06992
06993
06994
06995
06996
06997
06998
06999
07000
07001
07002
07003
07004
07005
07006
07007
07008
07009
07010
07011
07012
07013
07014
07015
07016
07017
07018
07019
07020
07021
07022
07023
07024
07025
07026
07027
07028
07029
07030
07031
07032
07033
07034
07035
07036
07037
07038
07039
07040
07041
07042
07043
07044
07045
07046
07047
07048
07049
07050
07051
07052
07053
07054
07055
07056
07057
07058
07059
07060
07061
07062
07063
07064
07065
07066
07067
07068
07069
07070
07071
07072
07073
07074
07075
07076
07077
07078
07079
07080
07081
07082
07083
07084
07085
07086
07087
07088
07089
07090
07091
07092
07093
07094
07095
07096
07097
07098
07099
07100
07101
07102
07103
07104
07105
07106
07107
07108
07109
07110
07111
07112
07113
07114
07115
07116
07117
07118
07119
07120
07121
07122
07123
07124
07125
07126
07127
07128
07129
07130
07131
07132
07133
07134
07135
07136
07137
07138
07139
07140
07141
07142
07143
07144
07145
07146
07147
07148
07149
07150
07151
07152
07153
07154
07155
07156
07157
07158
07159
07160
07161
07162
07163
07164
07165
07166
07167
07168
07169
07170
07171
07172
07173
07174
07175
07176
07177
07178
07179
07180
07181
07182
07183
07184
07185
07186
07187
07188
07189
07190
07191
07192
07193
07194
07195
07196
07197
07198
07199
07200
07201
07202
07203
07204
07205
07206
07207
07208
07209
07210
07211
07212
07213
07214
07215
07216
07217
07218
07219
07220
07221
07222
07223
07224
07225
07226
07227
07228
07229
07230
07231
07232
07233
07234
07235
07236
07237
07238
07239
07240
07241
07242
07243
07244
07245
07246
07247
07248
07249
07250
07251
07252
07253
07254
07255
07256
07257
07258
07259
07260
07261
07262
07263
07264
07265
07266
07267
07268
07269
07270
07271
07272
07273
07274
07275
07276
07277
07278
07279
07280
07281
07282
07283
07284
07285
07286
07287
07288
07289
07290
07291
07292
07293
07294
07295
07296
07297
07298
07299
07300
07301
07302
07303
07304
07305
07306
07307
07308
07309
07310
07311
07312
07313
07314
07315
07316
07317
07318
07319
07320
07321
07322
07323
07324
07325
07326
07327
07328
07329
07330
07331
07332
07333
07334
07335
07336
07337
07338
07339
07340
07341
07342
07343
07344
07345
07346
07347
07348
07349
07350
07351
07352
07353
07354
07355
07356
07357
07358
07359
07360
07361
07362
07363
07364
07365
07366
07367
07368
07369
07370
07371
07372
07373
07374
07375
07376
07377
07378
07379
07380
07381
07382
07383
07384
07385
07386
07387
07388
07389
07390
07391
07392
07393
07394
07395
07396
07397
07398
07399
07400
07401
07402
07403
07404
07405
07406
07407
07408
07409
07410
07411
07412
07413
07414
07415
07416
07417
07418
07419
07420
07421
07422
07423
07424
07425
07426
07427
07428
07429
07430
07431
07432
07433
07434
07435
07436
07437
07438
07439
07440
07441
07442
07443
07444
07445
07446
07447
07448
07449
07450
07451
07452
07453
07454
07455
07456
07457
07458
07459
07460
07461
07462
07463
07464
07465
07466
07467
07468
07469
07470
07471
07472
07473
07474
07475
07476
07477
07478
07479
07480
07481
07482
07483
07484
07485
07486
07487
07488
07489
07490
07491
07492
07493
07494
07495
07496
07497
07498
07499
07500
07501
07502
07503
07504
07505
07506
07507
07508
07509
07510
07511
07512
07513
07514
07515
07516
07517
07518
07519
07520
07521
07522
07523
07524
07525
07526
07527
07528
07529
07530
07531
07532
07533
07534
07535
07536
07537
07538
07539
07540
07541
07542
07543
07544
07545
07546
07547
07548
07549
07550
07551
07552
07553
07554
07555
07556
07557
07558
07559
07560
07561
07562
07563
07564
07565
07566
07567
07568
07569
07570
07571
07572
07573
07574
07575
07576
07577
07578
07579
07580
07581
07582
07583
07584
07585
07586
07587
07588
07589
07590
07591
07592
07593
07594
07595
07596
07597
07598
07599
07600
07601
07602
07603
07604
07605
07606
07607
07608
07609
07610
07611
07612
07613
07614
07615
07616
07617
07618
07619
07620
07621
07622
07623
07624
07625
07626
07627
07628
07629
07630
07631
07632
07633
07634
07635
07636
07637
07638
07639
07640
07641
07642
07643
07644
07645
07646
07647
07648
07649
07650
07651
07652
07653
07654
07655
07656
07657
07658
07659
07660
07661
07662
07663
07664
07665
07666
07667
07668
07669
07670
07671
07672
07673
07674
07675
07676
07677
07678
07679
07680
07681
07682
07683
07684
07685
07686
07687
07688
07689
07690
07691
07692
07693
07694
07695
07696
07697
07698
07699
07700
07701
07702
07703
07704
07705
07706
07707
07708
07709
07710
07711
07712
07713
07714
07715
07716
07717
07718
07719
07720
07721
07722
07723
07724
07725
07726
07727
07728
07729
07730
07731
07732
07733
07734
07735
07736
07737
07738
07739
07740
07741
07742
07743
07744
07745
07746
07747
07748
07749
07750
07751
07752
07753
07754
07755
07756
07757
07758
07759
07760
07761
07762
07763
07764
07765
07766
07767
07768
07769
07770
07771
07772
07773
07774
07775
07776
07777
07778
07779
07780
07781
07782
07783
07784
07785
07786
07787
07788
07789
07790
07791
07792
07793
07794
07795
07796
07797
07798
07799
07800
07801
07802
07803
07804
07805
07806
07807
07808
07809
07810
07811
07812
07813
07814
07815
07816
07817
07818
07819
07820
07821
07822
07823
07824
07825
07826
07827
07828
07829
07830
07831
07832
07833
07834
07835
07836
07837
07838
07839
07840
07841
07842
07843
07844
07845
07846
07847
07848
07849
07850
07851
07852
07853
07854
07855
07856
07857
07858
07859
07860
07861
07862
07863
07864
07865
07866
07867
07868
07869
07870
07871
07872
07873
07874
07875
07876
07877
07878
07879
07880
07881
07882
07883
07884
07885
07886
07887
07888
07889
07890
07891
07892
07893
07894
07895
07896
07897
07898
07899
07900
07901
07902
07903
07904
07905
07906
07907
07908
07909
07910
07911
07912
07913
07914
07915
07916
07917
07918
07919
07920
07921
07922
07923
07924
07925
07926
07927
07928
07929
07930
07931
07932
07933
07934
07935
07936
07937
07938
07939
07940
07941
07942
07943
07944
07945
07946
07947
07948
07949
07950
07951
07952
07953
07954
07955
07956
07957
07958
07959
07960
07961
07962
07963
07964
07965
07966
07967
07968
07969
07970
07971
07972
07973
07974
07975
07976
07977
07978
07979
07980
07981
07982
07983
07984
07985
07986
07987
07988
07989
07990
07991
07992
07993
07994
07995
07996
07997
07998
07999
08000
08001
08002
08003
08004
08005
08006
08007
08008
08009
08010
08011
08012
08013
08014
08015
08016
08017
08018
08019
08020
08021
08022
08023
08024
08025
08026
08027
08028
08029
08030
08031
08032
08033
08034
08035
08036
08037
08038
08039
08040
08041
08042
08043
08044
08045
08046
08047
08048
08049
08050
08051
08052
08053
08054
08055
08056
08057
08058
08059
08060
08061
08062
08063
08064
08065
08066
08067
08068
08069
08070
08071
08072
08073
08074
08075
08076
08077
08078
08079
08080
08081
08082
08083
08084
08085
08086
08087
08088
08089
08090
08091
08092
08093
08094
08095
08096
08097
08098
08099
08099
08100
08101
08102
08103
08104
08105
08106
08107
08108
08109
08110
08111
08112
08113
08114
08115
08116
08117
08118
08119
08120
08121
08122
08123
08124
08125
08126
08127
08128
08129
08130
08131
08132
08133
08134
08135
08136
08137
08138
08139
08140
08141
08142
08143
08144
08145
08146
08147
08148
08149
08150
08151
08152
08153
08154
08155
08156
08157
08158
08159
08160
08161
08162
08163
08164
08165
08166
08167
08168
08169
08170
08171
08172
08173
08174
08175
08176
08177
08178
08179
08180
08181
08182
08183
08184
08185
08186
08187
08188
08189
08190
08191
08192
08193
08194
08195
08196
08197
08198
08199
08199
08200
08201
08202
08203
08204
08205
08206
08207
08208
08209
08210
08211
08212
08213
08214
08215
08216
08217
08218
08219
08220
08221
08222
08223
08224
08225
08226
08227
08228
08229
08229
08230
08231
08232
08233
08234
08235
08236
08237
08238
08239
08239
08240
08241
08242
08243
08244
08245
08246
08247
08248
08249
08249
08250
08251
08252
08253
08254
08255
08256
08257
08258
08259
08259
08260
08261
08262
08263
08264
08265
08266
08267
08268
08269
08269
08270
08271
08272
08273
08274
08275
08276
08277
08278
08279
08279
08280
08281
08282
08283
08284
08285
08286
08287
08288
08289
08289
08290
08291
08292
08293
08294
08295
08296
08297
08298
08299
08299
08300
08301
08302
08303
08304
08305
08306
08307
08308
08309
08309
08310
08311
08312
08313
08314
08315
08316
08317
08318
08319
08319
08320
08321
08322
08323
08324
08325
08326
08327
08328
08329
08329
08330
08331
08332
08333
08334
08335
08336
08337
08338
08339
08339
08340
08341
08342
08343
08344
08345
08346
08347
08348
08349
08349
08350
08351
08352
08353
08354
08355
08356
08357
08358
08359
08359
08360
08361
08362
08363
08364
08365
08366
08367
08368
08369
08369
08370
08371
08372
08373
08374
08375
08376
08377
08378
08379
08379
08380
08381
08382
08383
08384
08385
08386
08387
08388
08389
08389
08390
08391
08392
08393
08394
08395
08396
08397
08398
08399
08399
08400
08401
08402
08403
08404
08405
08406
08407
08408
08409
08409
08410
08411
08412
08413
08414
08415
08416
08417
08418
08419
08419
08420
08421
08422
08423
08424
08425
08426
08427
08428
08429
08429
08430
08431
08432
08433
08434
08435
08436
08437
08438
08439
08439
08440
08441
08442
08443
08444
08445
08446
08447
08448
08449
08449
08450
08451
08452
08453
08454
08455
08456
08457
08458
08459
08459
08460
08461
08462
08463
08464
08465
08466
08467
08468
08469
08469
08470
08471
08472
08473
08474
08475
08476
08477
08478
08479
08479
08480
08481
08482
08483
08484
08485
08486
08487
08488
08489
08489
08490
08491
08492
08493
08494
08495
08496
08497
08498
08499
08499
08500
08501
08502
08503
08504
08505
08506
08507
08508
08509
08509
08510
08511
08512
08513
08514
08515
08516
08517
08518
08519
08519
08520
08521
08522
08523
08524
08525
08526
08527
08528
08529
08529
08530
08531
08532
0853
```

```
06909         return true;
06910     }
06911 }
06912 }
06913 }
06914 }
06915 return false;
06916 }
06917
06918 inline socket_t
06919 Server::create_server_socket(const std::string &host, int port,
06920                               int socket_flags,
06921                               SocketOptions socket_options) const {
06922     return detail::create_socket(
06923         host, std::string(), port, address_family_, socket_flags, tcp_nodelay_,
06924         ipv6_v6only_, std::move(socket_options),
06925         [](socket_t sock, struct addrinfo &ai, bool & /*quit*/) -> bool {
06926             if (::bind(sock, ai.ai_addr, static_cast<socklen_t>(ai.ai_addrlen))) {
06927                 return false;
06928             }
06929             if (::listen(sock, CPPHTTPLIB_LISTEN_BACKLOG)) { return false; }
06930             return true;
06931         });
06932 }
06933
06934 inline int Server::bind_internal(const std::string &host, int port,
06935                                   int socket_flags) {
06936     if (is_decommissioned) { return -1; }
06937
06938     if (!lis_valid()) { return -1; }
06939
06940     svr_sock_ = create_server_socket(host, port, socket_flags, socket_options_);
06941     if (svr_sock_ == INVALID_SOCKET) { return -1; }
06942
06943     if (port == 0) {
06944         struct sockaddr_storage addr;
06945         socklen_t addr_len = sizeof(addr);
06946         if (getsockname(svr_sock_, reinterpret_cast<struct sockaddr*>(&addr),
06947                         &addr_len) == -1) {
06948             return -1;
06949         }
06950         if (addr.ss_family == AF_INET) {
06951             return ntohs(reinterpret_cast<struct sockaddr_in*>(&addr)->sin_port);
06952         } else if (addr.ss_family == AF_INET6) {
06953             return ntohs(reinterpret_cast<struct sockaddr_in6*>(&addr)->sin6_port);
06954         } else {
06955             return -1;
06956         }
06957     } else {
06958         return port;
06959     }
06960 }
06961
06962 inline bool Server::listen_internal() {
06963     if (is_decommissioned) { return false; }
06964
06965     auto ret = true;
06966     is_running_ = true;
06967     auto se = detail::scope_exit([&]{} { is_running_ = false; });
06968
06969     {
06970         std::unique_ptr<TaskQueue> task_queue(new_task_queue());
06971
06972         while (svr_sock_ != INVALID_SOCKET) {
06973 #ifndef _WIN32
06974             if (idle_interval_sec_ > 0 || idle_interval_usec_ > 0) {
06975 #endif
06976                 auto val = detail::select_read(svr_sock_, idle_interval_sec_,
06977                                         idle_interval_usec_);
06978                 if (val == 0) { // Timeout
06979                     task_queue->on_idle();
06980                     continue;
06981                 }
06982 #ifndef _WIN32
06983             }
06984 #endif
06985
06986 #if defined _WIN32
06987     // sockets connected via WSAAccept inherit flags NO_HANDLE_INHERIT,
06988     // OVERLAPPED
06989     socket_t sock = WSAAccept(svr_sock_, nullptr, nullptr, nullptr, 0);
06990 #elif defined SOCK_CLOEXEC
06991     socket_t sock = accept4(svr_sock_, nullptr, nullptr, SOCK_CLOEXEC);
06992 #else
06993     socket_t sock = accept(svr_sock_, nullptr, nullptr);
06994 #endif
06995 }
```



```
07083     }
07084   }
07085 }
07086
07087 // Read content into `req.body`
07088 if (!read_content(strm, req, res)) { return false; }
07089 }
07090
07091 // Regular handler
07092 if (req.method == "GET" || req.method == "HEAD") {
07093   return dispatch_request(req, res, get_handlers_);
07094 } else if (req.method == "POST") {
07095   return dispatch_request(req, res, post_handlers_);
07096 } else if (req.method == "PUT") {
07097   return dispatch_request(req, res, put_handlers_);
07098 } else if (req.method == "DELETE") {
07099   return dispatch_request(req, res, delete_handlers_);
07100 } else if (req.method == "OPTIONS") {
07101   return dispatch_request(req, res, options_handlers_);
07102 } else if (req.method == "PATCH") {
07103   return dispatch_request(req, res, patch_handlers_);
07104 }
07105
07106 res.status = StatusCode::BadRequest_400;
07107 return false;
07108 }
07109
07110 inline bool Server::dispatch_request(Request &req, Response &res,
07111                           const Handlers &handlers) const {
07112   for (const auto &x : handlers) {
07113     const auto &matcher = x.first;
07114     const auto &handler = x.second;
07115
07116     if (matcher->match(req)) {
07117       handler(req, res);
07118       return true;
07119     }
07120   }
07121   return false;
07122 }
07123
07124 inline void Server::apply_ranges(const Request &req, Response &res,
07125                                   std::string &content_type,
07126                                   std::string &boundary) const {
07127   if (req.ranges.size() > 1 && res.status == StatusCode::PartialContent_206) {
07128     auto it = res.headers.find("Content-Type");
07129     if (it != res.headers.end()) {
07130       content_type = it->second;
07131       res.headers.erase(it);
07132     }
07133
07134     boundary = detail::make_multipart_data_boundary();
07135
07136     res.set_header("Content-Type",
07137                   "multipart/byteranges; boundary=" + boundary);
07138   }
07139
07140   auto type = detail::encoding_type(req, res);
07141
07142   if (res.body.empty()) {
07143     if (res.content_length_ > 0) {
07144       size_t length = 0;
07145       if (req.ranges.empty() || res.status != StatusCode::PartialContent_206) {
07146         length = res.content_length_;
07147       } else if (req.ranges.size() == 1) {
07148         auto offset_and_length = detail::get_range_offset_and_length(
07149           req.ranges[0], res.content_length_);
07150
07151         length = offset_and_length.second;
07152
07153         auto content_range = detail::make_content_range_header_field(
07154           offset_and_length, res.content_length_);
07155         res.set_header("Content-Range", content_range);
07156       } else {
07157         length = detail::get_multipart_ranges_data_length(
07158           req, boundary, content_type, res.content_length_);
07159       }
07160       res.set_header("Content-Length", std::to_string(length));
07161     } else {
07162       if (res.content_provider_) {
07163         if (res.is_chunked_content_provider_) {
07164           res.set_header("Transfer-Encoding", "chunked");
07165           if (type == detail::EncodingType::Gzip) {
07166             res.set_header("Content-Encoding", "gzip");
07167           } else if (type == detail::EncodingType::Brotli) {
07168             res.set_header("Content-Encoding", "br");
07169           } else if (type == detail::EncodingType::Zstd) {
```

```

07170         res.set_header("Content-Encoding", "zstd");
07171     }
07172   }
07173 }
07174 }
07175 } else {
07176   if (req.ranges.empty() || res.status != StatusCode::PartialContent_206) {
07177   ;
07178 } else if (req.ranges.size() == 1) {
07179   auto offset_and_length =
07200     detail::get_range_offset_and_length(req.ranges[0], res.body.size());
07201   auto offset = offset_and_length.first;
07202   auto length = offset_and_length.second;
07203
07204   auto content_range = detail::make_content_range_header_field(
07205     offset_and_length, res.body.size());
07206   res.set_header("Content-Range", content_range);
07207
07208   assert(offset + length <= res.body.size());
07209   res.body = res.body.substr(offset, length);
07210 } else {
07211   std::string data;
07212   detail::make_multipart_ranges_data(req, res, boundary, content_type,
07213                                       res.body.size(), data);
07214   res.body.swap(data);
07215 }
07216
07217 if (type != detail::EncodingType::None) {
07218   std::unique_ptr<detail::compressor> compressor;
07219   std::string content_encoding;
07220
07221   if (type == detail::EncodingType::Gzip) {
07222 #ifdef CPPHTTPLIB_ZLIB_SUPPORT
07223     compressor = detail::make_unique<detail::gzip_compressor>();
07224     content_encoding = "gzip";
07225 #endif
07226   } else if (type == detail::EncodingType::Brotli) {
07227 #ifdef CPPHTTPLIB_BROTLI_SUPPORT
07228     compressor = detail::make_unique<detail::brotli_compressor>();
07229     content_encoding = "br";
07230 #endif
07231   } else if (type == detail::EncodingType::Zstd) {
07232 #ifdef CPPHTTPLIB_ZSTD_SUPPORT
07233     compressor = detail::make_unique<detail::zstd_compressor>();
07234     content_encoding = "zstd";
07235 #endif
07236   }
07237
07238   if (compressor->compress(res.body.data(), res.body.size(), true,
07239     [&](const char *data, size_t data_len) {
07240       compressed.append(data, data_len);
07241       return true;
07242     })) {
07243     res.body.swap(compressed);
07244     res.set_header("Content-Encoding", content_encoding);
07245   }
07246 }
07247
07248 auto length = std::to_string(res.body.size());
07249 res.set_header("Content-Length", length);
07250
07251 inline bool Server::dispatch_request_for_content_reader(
07252   Request &req, Response &res, ContentReader content_reader,
07253   const HandlersForContentReader &handlers) const {
07254   for (const auto &x : handlers) {
07255     const auto &matcher = x.first;
07256     const auto &handler = x.second;
07257
07258     if (matcher->match(req)) {
07259       handler(req, res, content_reader);
07260       return true;
07261     }
07262   }
07263   return false;
07264 }
07265
07266 inline bool
07267 Server::process_request(Stream &strm, const std::string &remote_addr,
07268                           int remote_port, const std::string &local_addr,
07269                           int local_port, bool close_connection,
07270                           bool &connection_closed,
07271                           const std::function<void(Request &)> &setup_request) {

```

```
07257 std::array<char, 2048> buf{};
07258 detail::stream_line_reader line_reader(strm, buf.data(), buf.size());
07260
07261 // Connection has been closed on client
07262 if (!line_reader.getline()) { return false; }
07263
07264 Request req;
07265
07266 Response res;
07267 res.version = "HTTP/1.1";
07268 res.headers = default_headers_;
07269
07270 // Request line and headers
07271 if (!parse_request_line(line_reader.ptr(), req) ||
07272 !detail::read_headers(strm, req.headers)) {
07273 res.status = StatusCode::BadRequest_400;
07274 return write_response(strm, close_connection, req, res);
07275 }
07276
07277 // Check if the request URI doesn't exceed the limit
07278 if (req.target.size() > CPPHTTPLIB_REQUEST_URI_MAX_LENGTH) {
07279 Headers dummy;
07280 detail::read_headers(strm, dummy);
07281 res.status = StatusCode::UriTooLong_414;
07282 return write_response(strm, close_connection, req, res);
07283 }
07284
07285 if (req.get_header_value("Connection") == "close") {
07286 connection_closed = true;
07287 }
07288
07289 if (req.version == "HTTP/1.0" &&
07290 req.get_header_value("Connection") != "Keep-Alive") {
07291 connection_closed = true;
07292 }
07293
07294 req.remote_addr = remote_addr;
07295 req.remote_port = remote_port;
07296 req.set_header("REMOTE_ADDR", req.remote_addr);
07297 req.set_header("REMOTE_PORT", std::to_string(req.remote_port));
07298
07299 req.local_addr = local_addr;
07300 req.local_port = local_port;
07301 req.set_header("LOCAL_ADDR", req.local_addr);
07302 req.set_header("LOCAL_PORT", std::to_string(req.local_port));
07303
07304 if (req.has_header("Range")) {
07305 const auto &range_header_value = req.get_header_value("Range");
07306 if (!detail::parse_range_header(range_header_value, req.ranges)) {
07307 res.status = StatusCode::RangeNotSatisfiable_416;
07308 return write_response(strm, close_connection, req, res);
07309 }
07310 }
07311
07312 if (setup_request) { setup_request(req); }
07313
07314 if (req.get_header_value("Expect") == "100-continue") {
07315 int status = StatusCode::Continue_100;
07316 if (expect_100_continue_handler_) {
07317 status = expect_100_continue_handler_(req, res);
07318 }
07319 switch (status) {
07320 case StatusCode::Continue_100:
07321 case StatusCode::ExpectationFailed_417:
07322 detail::write_response_line(strm, status);
07323 strm.write("\r\n");
07324 break;
07325 default:
07326 connection_closed = true;
07327 return write_response(strm, true, req, res);
07328 }
07329 }
07330
07331 // Setup `is_connection_closed` method
07332 req.is_connection_closed = [&]() {
07333 return !detail::is_socket_alive(strm.socket());
07334 };
07335
07336 // Routing
07337 auto routed = false;
07338 #ifndef CPPHTTPLIB_NO_EXCEPTIONS
07339 routed = routing(req, res, strm);
07340 #else
07341 try {
07342 routed = routing(req, res, strm);
07343 } catch (std::exception &e) {
```

```
07344     if (exception_handler_) {
07345         auto ep = std::current_exception();
07346         exception_handler_(req, res, ep);
07347         routed = true;
07348     } else {
07349         res.status = StatusCode::InternalServerError_500;
07350         std::string val;
07351         auto s = e.what();
07352         for (size_t i = 0; s[i]; i++) {
07353             switch (s[i]) {
07354                 case '\r': val += "\r"; break;
07355                 case '\n': val += "\n"; break;
07356                 default: val += s[i]; break;
07357             }
07358         }
07359         res.set_header("EXCEPTION_WHAT", val);
07360     }
07361 } catch (...) {
07362     if (exception_handler_) {
07363         auto ep = std::current_exception();
07364         exception_handler_(req, res, ep);
07365         routed = true;
07366     } else {
07367         res.status = StatusCode::InternalServerError_500;
07368         res.set_header("EXCEPTION_WHAT", "UNKNOWN");
07369     }
07370 }
07371 #endif
07372 if (routed) {
07373     if (res.status == -1) {
07374         res.status = req.ranges.empty() ? StatusCode::OK_200
07375                               : StatusCode::PartialContent_206;
07376     }
07377
07378 // Serve file content by using a content provider
07379 if (!res.file_content_path_.empty()) {
07380     const auto &path = res.file_content_path_;
07381     auto mm = std::make_shared<detail::mmap>(path.c_str());
07382     if (!mm->is_open()) {
07383         res.body.clear();
07384         res.content_length_ = 0;
07385         res.content_provider_ = nullptr;
07386         res.status = StatusCode::NotFound_404;
07387         return write_response(strm, close_connection, req, res);
07388     }
07389
07390     auto content_type = res.file_content_content_type_;
07391     if (content_type.empty()) {
07392         content_type = detail::find_content_type(
07393             path, file_extension_and_mimetype_map_, default_file_mimetype_);
07394     }
07395
07396     res.set_content_provider(
07397         mm->size(), content_type,
07398         [mm](size_t offset, size_t length, DataSink &sink) -> bool {
07399             sink.write(mm->data() + offset, length);
07400             return true;
07401         });
07402     }
07403
07404     if (detail::range_error(req, res)) {
07405         res.body.clear();
07406         res.content_length_ = 0;
07407         res.content_provider_ = nullptr;
07408         res.status = StatusCode::RangeNotSatisfiable_416;
07409         return write_response(strm, close_connection, req, res);
07410     }
07411
07412     return write_response_with_content(strm, close_connection, req, res);
07413 } else {
07414     if (res.status == -1) { res.status = StatusCode::NotFound_404; }
07415
07416     return write_response(strm, close_connection, req, res);
07417 }
07418 }
07419
07420 inline bool Server::is_valid() const { return true; }
07421
07422 inline bool Server::process_and_close_socket(socket_t sock) {
07423     std::string remote_addr;
07424     int remote_port = 0;
07425     detail::get_remote_ip_and_port(sock, remote_addr, remote_port);
07426
07427     std::string local_addr;
07428     int local_port = 0;
07429     detail::get_local_ip_and_port(sock, local_addr, local_port);
07430 }
```

```

07431 auto ret = detail::process_server_socket(
07432     svr_sock_, sock, keep_alive_max_count_, keep_alive_timeout_sec_,
07433     read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
07434     write_timeout_usec_,
07435     [&](Stream &strm, bool close_connection, bool &connection_closed) {
07436         return process_request(strm, remote_addr, remote_port, local_addr,
07437             local_port, close_connection, connection_closed,
07438             nullptr);
07439     });
07440
07441 detail::shutdown_socket(sock);
07442 detail::close_socket(sock);
07443 return ret;
07444 }
07445
07446 // HTTP client implementation
07447 inline ClientImpl::ClientImpl(const std::string &host)
07448 : ClientImpl(host, 80, std::string(), std::string()) {}
07449
07450 inline ClientImpl::ClientImpl(const std::string &host, int port)
07451 : ClientImpl(host, port, std::string(), std::string()) {}
07452
07453 inline ClientImpl::ClientImpl(const std::string &host, int port,
07454     const std::string &client_cert_path,
07455     const std::string &client_key_path)
07456 : host_(detail::escape_abstract_namespace_unix_domain(host)), port_(port),
07457     host_and_port_(adjust_host_string(host_) + ":" + std::to_string(port)),
07458     client_cert_path_(client_cert_path), client_key_path_(client_key_path) {}
07459
07460 inline ClientImpl::~ClientImpl() {
07461     // Wait until all the requests in flight are handled.
07462     size_t retry_count = 10;
07463     while (retry_count-- > 0) {
07464         {
07465             std::lock_guard<std::mutex> guard(socket_mutex_);
07466             if (socket_requests_in_flight_ == 0) { break; }
07467         }
07468         std::this_thread::sleep_for(std::chrono::milliseconds{1});
07469     }
07470
07471     std::lock_guard<std::mutex> guard(socket_mutex_);
07472     shutdown_socket(socket_);
07473     close_socket(socket_);
07474 }
07475
07476 inline bool ClientImpl::is_valid() const { return true; }
07477
07478 inline void ClientImpl::copy_settings(const ClientImpl &rhs) {
07479     client_cert_path_ = rhs.client_cert_path_;
07480     client_key_path_ = rhs.client_key_path_;
07481     connection_timeout_sec_ = rhs.connection_timeout_sec_;
07482     read_timeout_sec_ = rhs.read_timeout_sec_;
07483     read_timeout_usec_ = rhs.read_timeout_usec_;
07484     write_timeout_sec_ = rhs.write_timeout_sec_;
07485     write_timeout_usec_ = rhs.write_timeout_usec_;
07486     max_timeout_msec_ = rhs.max_timeout_msec_;
07487     basic_auth_username_ = rhs.basic_auth_username_;
07488     basic_auth_password_ = rhs.basic_auth_password_;
07489     bearer_token_auth_token_ = rhs.bearer_token_auth_token_;
07490 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07491     digest_auth_username_ = rhs.digest_auth_username_;
07492     digest_auth_password_ = rhs.digest_auth_password_;
07493 #endif
07494     keep_alive_ = rhs.keep_alive_;
07495     follow_location_ = rhs.follow_location_;
07496     url_encode_ = rhs.url_encode_;
07497     address_family_ = rhs.address_family_;
07498     tcp_nodelay_ = rhs.tcp_nodelay_;
07499     ipv6_v6only_ = rhs.ipv6_v6only_;
07500     socket_options_ = rhs.socket_options_;
07501     compress_ = rhs.compress_;
07502     decompress_ = rhs.decompress_;
07503     interface_ = rhs.interface_;
07504     proxy_host_ = rhs.proxy_host_;
07505     proxy_port_ = rhs.proxy_port_;
07506     proxy_basic_auth_username_ = rhs.proxy_basic_auth_username_;
07507     proxy_basic_auth_password_ = rhs.proxy_basic_auth_password_;
07508     proxy_bearer_token_auth_token_ = rhs.proxy_bearer_token_auth_token_;
07509 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07510     proxy_digest_auth_username_ = rhs.proxy_digest_auth_username_;
07511     proxy_digest_auth_password_ = rhs.proxy_digest_auth_password_;
07512 #endif
07513 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07514     ca_cert_file_path_ = rhs.ca_cert_file_path_;
07515     ca_cert_dir_path_ = rhs.ca_cert_dir_path_;
07516     ca_cert_store_ = rhs.ca_cert_store_;
07517 #endif

```

```

07518 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07519     server_certificate_verification_ = rhs.server_certificate_verification_;
07520     server_hostname_verification_ = rhs.server_hostname_verification_;
07521     server_certificate_verifier_ = rhs.server_certificate_verifier_;
07522 #endif
07523     logger_ = rhs.logger_;
07524 }
07525
07526 inline socket_t ClientImpl::create_client_socket(Error &error) const {
07527     if (!proxy_host_.empty() && proxy_port_ != -1) {
07528         return detail::create_client_socket(
07529             proxy_host_, std::string(), proxy_port_, address_family_, tcp_nodelay_,
07530             ipv6_v6only_, socket_options_, connection_timeout_sec_,
07531             connection_timeout_usec_, read_timeout_sec_, read_timeout_usec_,
07532             write_timeout_sec_, write_timeout_usec_, interface_, error);
07533     }
07534
07535     // Check is custom IP specified for host_
07536     std::string ip;
07537     auto it = addr_map_.find(host_);
07538     if (it != addr_map_.end()) { ip = it->second; }
07539
07540     return detail::create_client_socket(
07541         host_, ip, port_, address_family_, tcp_nodelay_, ipv6_v6only_,
07542         socket_options_, connection_timeout_sec_, connection_timeout_usec_,
07543         read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
07544         write_timeout_usec_, interface_, error);
07545 }
07546
07547 inline bool ClientImpl::create_and_connect_socket(Socket &socket,
07548                                     Error &error) {
07549     auto sock = create_client_socket(error);
07550     if (sock == INVALID_SOCKET) { return false; }
07551     socket.sock = sock;
07552     return true;
07553 }
07554
07555 inline void ClientImpl::shutdown_ssl(Socket & /*socket*/,
07556                                         bool /*shutdown_gracefully*/) {
07557     // If there are any requests in flight from threads other than us, then it's
07558     // a thread-unsafe race because individual ssl* objects are not thread-safe.
07559     assert(socket_requests_in_flight_ == 0 ||
07560           socket_requests_are_from_thread_ == std::this_thread::get_id());
07561 }
07562
07563 inline void ClientImpl::shutdown_socket(Socket &socket) const {
07564     if (socket.sock == INVALID_SOCKET) { return; }
07565     detail::shutdown_socket(socket.sock);
07566 }
07567
07568 inline void ClientImpl::close_socket(Socket &socket) {
07569     // If there are requests in flight in another thread, usually closing
07570     // the socket will be fine and they will simply receive an error when
07571     // using the closed socket, but it is still a bug since rarely the OS
07572     // may reassign the socket id to be used for a new socket, and then
07573     // suddenly they will be operating on a live socket that is different
07574     // than the one they intended!
07575     assert(socket_requests_in_flight_ == 0 ||
07576           socket_requests_are_from_thread_ == std::this_thread::get_id());
07577
07578     // It is also a bug if this happens while SSL is still active
07579 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07580     assert(socket.ssl == nullptr);
07581 #endif
07582     if (socket.sock == INVALID_SOCKET) { return; }
07583     detail::close_socket(socket.sock);
07584     socket.sock = INVALID_SOCKET;
07585 }
07586
07587 inline bool ClientImpl::read_response_line(Stream &strm, const Request &req,
07588                                               Response &res) const {
07589     std::array<char, 2048> buf{};
07590
07591     detail::stream_line_reader line_reader(strm, buf.data(), buf.size());
07592
07593     if (!line_reader.getline()) { return false; }
07594
07595 #ifdef CPPHTTPLIB_ALLOW_LF_AS_LINE_TERMINATOR
07596     thread_local const std::regex re("(HTTP/1\\.[01]) (\\d{3})(?:(.*)?)\\r?\\n");
07597 #else
07598     thread_local const std::regex re("(HTTP/1\\.[01]) (\\d{3})(?:(.*)?)\\r\\n");
07599 #endif
07600
07601     std::cmatch m;
07602     if (!std::regex_match(line_reader.ptr(), m, re)) {
07603         return req.method == "CONNECT";
07604     }

```

```

07605     res.version = std::string(m[1]);
07606     res.status = std::stoi(std::string(m[2]));
07607     res.reason = std::string(m[3]);
07608
07609     // Ignore '100 Continue'
07610     while (res.status == StatusCode::Continue_100) {
07611         if (!line_reader.getline()) { return false; } // CRLF
07612         if (!line_reader.getline()) { return false; } // next response line
07613
07614         if (!std::regex_match(line_reader.ptr(), m, re)) { return false; }
07615         res.version = std::string(m[1]);
07616         res.status = std::stoi(std::string(m[2]));
07617         res.reason = std::string(m[3]);
07618     }
07619
07620     return true;
07621 }
07622
07623 inline bool ClientImpl::send(Request &req, Response &res, Error &error) {
07624     std::lock_guard<std::recursive_mutex> request_mutex_guard(request_mutex_);
07625     auto ret = send_(req, res, error);
07626     if (error == Error::SSLPeerCouldBeClosed_) {
07627         assert(!ret);
07628         ret = send_(req, res, error);
07629     }
07630     return ret;
07631 }
07632
07633 inline bool ClientImpl::send_(Request &req, Response &res, Error &error) {
07634 {
07635     std::lock_guard<std::mutex> guard(socket_mutex_);
07636
07637     // Set this to false immediately - if it ever gets set to true by the end of
07638     // the request, we know another thread instructed us to close the socket.
07639     socket_should_be_closed_when_request_is_done_ = false;
07640
07641     auto is_alive = false;
07642     if (socket_.is_open()) {
07643         is_alive = detail::is_socket_alive(socket_.sock);
07644
07645 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07646         if (is_alive && is_ssl()) {
07647             if (detail::is_ssl_peer_could_be_closed(socket_.ssl, socket_.sock)) {
07648                 is_alive = false;
07649             }
07650         }
07651 #endif
07652
07653     if (!is_alive) {
07654         // Attempt to avoid sigpipe by shutting down non-gracefully if it seems
07655         // like the other side has already closed the connection. Also, there
07656         // cannot be any requests in flight from other threads since we locked
07657         // request_mutex_, so safe to close everything immediately
07658         const bool shutdown_gracefully = false;
07659         shutdown_ssl(socket_, shutdown_gracefully);
07660         shutdown_socket(socket_);
07661         close_socket(socket_);
07662     }
07663 }
07664
07665     if (!is_alive) {
07666         if (!create_and_connect_socket(socket_, error)) { return false; }
07667
07668 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07669     // TODO: refactoring
07670     if (is_ssl()) {
07671         auto &scli = static_cast<SSLClient &>(*this);
07672         if (!proxy_host_.empty() && proxy_port_ != -1) {
07673             auto success = false;
07674             if (!scli.connect_with_proxy(socket_, req.start_time_, res, success,
07675                                         error)) {
07676                 return success;
07677             }
07678         }
07679
07680         if (!scli.initialize_ssl(socket_, error)) { return false; }
07681     }
07682 #endif
07683 }
07684
07685     // Mark the current socket as being in use so that it cannot be closed by
07686     // anyone else while this request is ongoing, even though we will be
07687     // releasing the mutex.
07688     if (socket_requests_in_flight_ > 1) {
07689         assert(socket_requests_are_from_thread_ == std::this_thread::get_id());
07690     }
07691     socket_requests_in_flight_ += 1;

```

```

07692     socket_requests_are_from_thread_ = std::this_thread::get_id();
07693 }
07694
07695     for (const auto &header : default_headers_) {
07696         if (req.headers.find(header.first) == req.headers.end()) {
07697             req.headers.insert(header);
07698         }
07699     }
07700
07701     auto ret = false;
07702     auto close_connection = !keep_alive_;
07703
07704     auto se = detail::scope_exit([&]{
07705         // Briefly lock mutex in order to mark that a request is no longer ongoing
07706         std::lock_guard<std::mutex> guard(socket_mutex_);
07707         socket_requests_in_flight -= 1;
07708         if (socket_requests_in_flight <= 0) {
07709             assert(socket_requests_in_flight == 0);
07710             socket_requests_are_from_thread_ = std::thread::id();
07711         }
07712
07713         if (socket_should_be_closed_when_request_is_done_ || close_connection ||
07714             !ret) {
07715             shutdown_ssl(socket_, true);
07716             shutdown_socket(socket_);
07717             close_socket(socket_);
07718         }
07719     });
07720
07721     ret = process_socket(socket_, req.start_time_, [&](Stream &strm) {
07722         return handle_request(strm, req, res, close_connection, error);
07723     });
07724
07725     if (!ret) {
07726         if (error == Error::Success) { error = Error::Unknown; }
07727     }
07728
07729     return ret;
07730 }
07731
07732 inline Result ClientImpl::send(const Request &req) {
07733     auto req2 = req;
07734     return send_(std::move(req2));
07735 }
07736
07737 inline Result ClientImpl::send_(Request &&req) {
07738     auto res = detail::make_unique<Response>();
07739     auto error = Error::Success;
07740     auto ret = send(req, *res, error);
07741     return Result{ret ? std::move(res) : nullptr, error, std::move(req.headers)};
07742 }
07743
07744 inline bool ClientImpl::handle_request(Stream &strm, Request &req,
07745                                         Response &res, bool close_connection,
07746                                         Error &error) {
07747     if (req.path.empty()) {
07748         error = Error::Connection;
07749         return false;
07750     }
07751
07752     auto req_save = req;
07753
07754     bool ret;
07755
07756     if (!is_ssl() && !proxy_host_.empty() && proxy_port_ != -1) {
07757         auto req2 = req;
07758         req2.path = "http://" + host_and_port_ + req.path;
07759         ret = process_request(strm, req2, res, close_connection, error);
07760         req = req2;
07761         req.path = req_save.path;
07762     } else {
07763         ret = process_request(strm, req, res, close_connection, error);
07764     }
07765
07766     if (!ret) { return false; }
07767
07768     if (res.get_header_value("Connection") == "close" ||
07769         (res.version == "HTTP/1.0" && res.reason != "Connection established")) {
07770         // TODO this requires a not-entirely-obvious chain of calls to be correct
07771         // for this to be safe.
07772
07773         // This is safe to call because handle_request is only called by send_
07774         // which locks the request mutex during the process. It would be a bug
07775         // to call it from a different thread since it's a thread-safety issue
07776         // to do these things to the socket if another thread is using the socket.
07777         std::lock_guard<std::mutex> guard(socket_mutex_);
07778         shutdown_ssl(socket_, true);

```

```

07779     shutdown_socket(socket_);
07780     close_socket(socket_);
07781 }
07782
07783 if (300 < res.status && res.status < 400 && follow_location_) {
07784     req = req_save;
07785     ret = redirect(req, res, error);
07786 }
07787
07788 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07789 if ((res.status == StatusCode::Unauthorized_401 ||
07790     res.status == StatusCode::ProxyAuthenticationRequired_407) &&
07791     req.authorization_count_ < 5) {
07792     auto is_proxy = res.status == StatusCode::ProxyAuthenticationRequired_407;
07793     const auto &username =
07794         is_proxy ? proxy_digest_auth_username_ : digest_auth_username_;
07795     const auto &password =
07796         is_proxy ? proxy_digest_auth_password_ : digest_auth_password_;
07797
07798 if (!username.empty() && !password.empty()) {
07799     std::map<std::string, std::string> auth;
07800     if (detail::parse_www_authenticate(res, auth, is_proxy)) {
07801         Request new_req = req;
07802         new_req.authorization_count_ += 1;
07803         new_req.headers.erase(is_proxy ? "Proxy-Authorization"
07804                               : "Authorization");
07805         new_req.headers.insert(detail::make_digest_authentication_header(
07806             req, auth, new_req.authorization_count_, detail::random_string(10),
07807             username, password, is_proxy));
07808
07809     Response new_res;
07810
07811     ret = send(new_req, new_res, error);
07812     if (ret) { res = new_res; }
07813 }
07814 }
07815 }
07816 #endif
07817
07818 return ret;
07819 }
07820
07821 inline bool ClientImpl::redirect(Request &req, Response &res, Error &error) {
07822 if (req.redirect_count_ == 0) {
07823     error = Error::ExceedRedirectCount;
07824     return false;
07825 }
07826
07827 auto location = res.get_header_value("location");
07828 if (location.empty()) { return false; }
07829
07830 thread_local const std::regex re(
07831     R"((?:https?):)?(?://([[:a-zA-F\d:]])|([:^/?#]+)(?:(\d+))?(?:[^?#]*)(\?[^\#]*)(?:\.*))?");
07832
07833 std::smatch m;
07834 if (!std::regex_match(location, m, re)) { return false; }
07835
07836 auto scheme = is_ssl() ? "https" : "http";
07837
07838 auto next_scheme = m[1].str();
07839 auto next_host = m[2].str();
07840 if (next_host.empty()) { next_host = m[3].str(); }
07841 auto port_str = m[4].str();
07842 auto next_path = m[5].str();
07843 auto next_query = m[6].str();
07844
07845 auto next_port = port_;
07846 if (!port_str.empty()) {
07847     next_port = std::stoi(port_str);
07848 } else if (!next_scheme.empty()) {
07849     next_port = next_scheme == "https" ? 443 : 80;
07850 }
07851
07852 if (next_scheme.empty()) { next_scheme = scheme; }
07853 if (next_host.empty()) { next_host = host_; }
07854 if (next_path.empty()) { next_path = "/"; }
07855
07856 auto path = detail::decode_url(next_path, true) + next_query;
07857
07858 if (next_scheme == scheme && next_host == host_ && next_port == port_) {
07859     return detail::redirect(*this, req, res, path, location, error);
07860 } else {
07861     if (next_scheme == "https") {
07862 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
07863         SSLClient cli(next_host, next_port);
07864         cli.copy_settings(*this);
07865         if (ca_cert_store_) { cli.set_ca_cert_store(ca_cert_store_); }

```

```
07866     return detail::redirect(cli, req, res, path, location, error);
07867 #else
07868     return false;
07869 #endiff
07870 } else {
07871     ClientImpl cli(next_host, next_port);
07872     cli.copy_settings(*this);
07873     return detail::redirect(cli, req, res, path, location, error);
07874 }
07875 }
07876 }
07877
07878 inline bool ClientImpl::write_content_with_provider(Stream &strm,
07879                                     const Request &req,
07880                                     Error &error) const {
07881     auto is_shutting_down = []() { return false; };
07882
07883     if (req.is_chunked_content_provider_) {
07884         // TODO: Broth support
07885         std::unique_ptr<detail::compressor> compressor;
07886 #ifdef CPPHTTPPLIB_ZLIB_SUPPORT
07887         if (compress_) {
07888             compressor = detail::make_unique<detail::gzip_compressor>();
07889         } else
07890 #endiff
07891         {
07892             compressor = detail::make_unique<detail::nocompressor>();
07893         }
07894
07895     return detail::write_content_chunked(strm, req.content_provider_,
07896                                         is_shutting_down, *compressor, error);
07897 } else {
07898     return detail::write_content(strm, req.content_provider_, 0,
07899                                 req.content_length_, is_shutting_down, error);
07900 }
07901 }
07902
07903 inline bool ClientImpl::write_request(Stream &strm, Request &req,
07904                                         bool close_connection, Error &error) {
07905     // Prepare additional headers
07906     if (close_connection) {
07907         if (!req.has_header("Connection")) {
07908             req.set_header("Connection", "close");
07909         }
07910     }
07911
07912     if (!req.has_header("Host")) {
07913         if (is_ssl()) {
07914             if (port_ == 443) {
07915                 req.set_header("Host", host_);
07916             } else {
07917                 req.set_header("Host", host_and_port_);
07918             }
07919         } else {
07920             if (port_ == 80) {
07921                 req.set_header("Host", host_);
07922             } else {
07923                 req.set_header("Host", host_and_port_);
07924             }
07925         }
07926     }
07927
07928     if (!req.has_header("Accept")) { req.set_header("Accept", "*/*"); }
07929
07930     if (!req.content_receiver) {
07931         if (!req.has_header("Accept-Encoding")) {
07932             std::string accept_encoding;
07933 #ifdef CPPHTTPPLIB_BROTLI_SUPPORT
07934             accept_encoding = "br";
07935 #endiff
07936 #ifdef CPPHTTPPLIB_ZLIB_SUPPORT
07937             if (!accept_encoding.empty()) { accept_encoding += ", "; }
07938             accept_encoding += "gzip, deflate";
07939 #endiff
07940 #ifdef CPPHTTPPLIB_ZSTD_SUPPORT
07941             if (!accept_encoding.empty()) { accept_encoding += ", "; }
07942             accept_encoding += "zstd";
07943 #endiff
07944             req.set_header("Accept-Encoding", accept_encoding);
07945         }
07946
07947 #ifndef CPPHTTPPLIB_NO_DEFAULT_USER_AGENT
07948     if (!req.has_header("User-Agent")) {
07949         auto agent = std::string("cpp-httplib/") + CPPHTTPPLIB_VERSION;
07950         req.set_header("User-Agent", agent);
07951     }
07952 #endiff
```

```

07953  };
07954  if (req.body.empty()) {
07955    if (req.content_provider_) {
07956      if (!req.is_chunked_content_provider_) {
07957        if (!req.has_header("Content-Length")) {
07958          auto length = std::to_string(req.content_length_);
07959          req.set_header("Content-Length", length);
07960        }
07961      }
07962    }
07963  } else {
07964    if (req.method == "POST" || req.method == "PUT" ||
07965        req.method == "PATCH") {
07966      req.set_header("Content-Length", "0");
07967    }
07968  }
07969 } else {
07970   if (!req.has_header("Content-Type")) {
07971     req.set_header("Content-Type", "text/plain");
07972   }
07973   if (!req.has_header("Content-Length")) {
07974     auto length = std::to_string(req.body.size());
07975     req.set_header("Content-Length", length);
07976   }
07977 }
07978 }
07979 if (!basic_auth_password_.empty() || !basic_auth_username_.empty()) {
07980   if (req.has_header("Authorization")) {
07981     req.headers.insert(make_basic_authentication_header(
07982       basic_auth_username_, basic_auth_password_, false));
07983   }
07984 }
07985 }
07986 if (!proxy_basic_auth_username_.empty() &&
07987   !proxy_basic_auth_password_.empty()) {
07988   if (req.has_header("Proxy-Authorization")) {
07989     req.headers.insert(make_basic_authentication_header(
07990       proxy_basic_auth_username_, proxy_basic_auth_password_, true));
07991   }
07992 }
07993 }
07994 if (!bearer_token_auth_token_.empty()) {
07995   if (!req.has_header("Authorization")) {
07996     req.headers.insert(make_bearer_token_authentication_header(
07997       bearer_token_auth_token_, false));
07998   }
07999 }
08000 }
08001
08002 if (!proxy_bearer_token_auth_token_.empty()) {
08003   if (!req.has_header("Proxy-Authorization")) {
08004     req.headers.insert(make_bearer_token_authentication_header(
08005       proxy_bearer_token_auth_token_, true));
08006   }
08007 }
08008
08009 // Request line and headers
08010 {
08011   detail::BufferStream bstrm;
08012
08013   const auto &path_with_query =
08014     req.params.empty() ? req.path
08015       : append_query_params(req.path, req.params);
08016
08017   const auto &path =
08018     url_encode_ ? detail::encode_url(path_with_query) : path_with_query;
08019
08020   detail::write_request_line(bstrm, req.method, path);
08021
08022   header_writer_(bstrm, req.headers);
08023
08024   // Flush buffer
08025   auto &data = bstrm.get_buffer();
08026   if (!detail::write_data(strm, data.data(), data.size())) {
08027     error = Error::Write;
08028     return false;
08029   }
08030 }
08031
08032 // Body
08033 if (req.body.empty()) {
08034   return write_content_with_provider(strm, req, error);
08035 }
08036
08037 if (!detail::write_data(strm, req.body.data(), req.body.size())) {
08038   error = Error::Write;
08039   return false;

```

```
08040  }
08041
08042  return true;
08043 }
08044
08045 inline std::unique_ptr<Response> ClientImpl::send_with_content_provider(
08046     Request &req, const char *body, size_t content_length,
08047     ContentProvider content_provider,
08048     ContentProviderWithoutLength content_provider_without_length,
08049     const std::string &content_type, Error &error) {
08050     if (!content_type.empty()) { req.set_header("Content-Type", content_type); }
08051
08052 #ifndef CPPHTTPLIB_ZLIB_SUPPORT
08053     if (compress_) { req.set_header("Content-Encoding", "gzip"); }
08054 #endif
08055
08056 #ifndef CPPHTTPLIB_ZLIB_SUPPORT
08057     if (compress_ && !content_provider_without_length) {
08058         // TODO: Brothi support
08059         detail::gzip_compressor compressor;
08060
08061         if (content_provider) {
08062             auto ok = true;
08063             size_t offset = 0;
08064             DataSink data_sink;
08065
08066             data_sink.write = [&](const char *data, size_t data_len) -> bool {
08067                 if (ok) {
08068                     auto last = offset + data_len == content_length;
08069
08070                     auto ret = compressor.compress(
08071                         data, data_len, last,
08072                         [&](const char *compressed_data, size_t compressed_data_len) {
08073                             req.body.append(compressed_data, compressed_data_len);
08074                             return true;
08075                         });
08076
08077                     if (ret) {
08078                         offset += data_len;
08079                     } else {
08080                         ok = false;
08081                     }
08082                 }
08083                 return ok;
08084             };
08085
08086             while (ok && offset < content_length) {
08087                 if (!content_provider(offset, content_length - offset, data_sink)) {
08088                     error = Error::Canceled;
08089                     return nullptr;
08090                 }
08091             }
08092         } else {
08093             if (!compressor.compress(body, content_length, true,
08094                             [&](const char *data, size_t data_len) {
08095                                 req.body.append(data, data_len);
08096                                 return true;
08097                             })) {
08098                 error = Error::Compression;
08099                 return nullptr;
08100             }
08101         }
08102     } else
08103 #endif
08104 {
08105     if (content_provider) {
08106         req.content_length_ = content_length;
08107         req.content_provider_ = std::move(content_provider);
08108         req.is_chunked_content_provider_ = false;
08109     } else if (content_provider_without_length) {
08110         req.content_length_ = 0;
08111         req.content_provider_ = detail::ContentProviderAdapter(
08112             std::move(content_provider_without_length));
08113         req.is_chunked_content_provider_ = true;
08114         req.set_header("Transfer-Encoding", "chunked");
08115     } else {
08116         req.body.assign(body, content_length);
08117     }
08118 }
08119
08120 auto res = detail::make_unique<Response>();
08121 return send(req, *res, error) ? std::move(res) : nullptr;
08122 }
08123
08124 inline Result ClientImpl::send_with_content_provider(
08125     const std::string &method, const std::string &path, const Headers &headers,
08126     const char *body, size_t content_length, ContentProvider content_provider,
```

```
08127     ContentProviderWithoutLength content_provider_without_length,
08128     const std::string &content_type, Progress progress) {
08129     Request req;
08130     req.method = method;
08131     req.headers = headers;
08132     req.path = path;
08133     req.progress = progress;
08134     if (max_timeout_msec_ > 0) {
08135         req.start_time_ = std::chrono::steady_clock::now();
08136     }
08137
08138     auto error = Error::Success;
08139
08140     auto res = send_with_content_provider(
08141         req, body, content_length, std::move(content_provider),
08142         std::move(content_provider_without_length), content_type, error);
08143
08144     return Result{std::move(res), error, std::move(req.headers)};
08145 }
08146
08147 inline std::string
08148 ClientImpl::adjust_host_string(const std::string &host) const {
08149     if (host.find(':') != std::string::npos) { return "[" + host + "]"; }
08150     return host;
08151 }
08152
08153 inline bool ClientImpl::process_request(Stream &strm, Request &req,
08154                                         Response &res, bool close_connection,
08155                                         Error &error) {
08156     // Send request
08157     if (!write_request(strm, req, close_connection, error)) { return false; }
08158
08159 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
08160     if (is_ssl()) {
08161         auto is_proxy_enabled = !proxy_host_.empty() && proxy_port_ != -1;
08162         if (!is_proxy_enabled) {
08163             if (detail::is_ssl_peer_could_be_closed(socket_.ssl, socket_.sock)) {
08164                 error = Error::SSLPeerCouldBeClosed_;
08165                 return false;
08166             }
08167         }
08168     }
08169 #endif
08170
08171     // Receive response and headers
08172     if (!read_response_line(strm, req, res) ||
08173         !detail::read_headers(strm, res.headers)) {
08174         error = Error::Read;
08175         return false;
08176     }
08177
08178     // Body
08179     if ((res.status != StatusCode::NoContent_204) && req.method != "HEAD" &&
08180         req.method != "CONNECT") {
08181         auto redirect = 300 < res.status && res.status < 400 &&
08182             res.status != StatusCode::NotModified_304 &&
08183             follow_location_;
08184
08185         if (req.response_handler && !redirect) {
08186             if (!req.response_handler(res)) {
08187                 error = Error::Canceled;
08188                 return false;
08189             }
08190         }
08191
08192         auto out =
08193             req.content_receiver
08194             ? static_cast<ContentReceiverWithProgress>(
08195                 [&](const char *buf, size_t n, uint64_t off, uint64_t len) {
08196                     if (redirect) { return true; }
08197                     auto ret = req.content_receiver(buf, n, off, len);
08198                     if (!ret) { error = Error::Canceled; }
08199                     return ret;
08200                 })
08201             : static_cast<ContentReceiverWithProgress>(
08202                 [&](const char *buf, size_t n, uint64_t /*off*/,
08203                     uint64_t /*len*/) {
08204                     assert(res.body.size() + n <= res.body.max_size());
08205                     res.body.append(buf, n);
08206                     return true;
08207                 });
08208
08209         auto progress = [&](uint64_t current, uint64_t total) {
08210             if (!req.progress || redirect) { return true; }
08211             auto ret = req.progress(current, total);
08212             if (!ret) { error = Error::Canceled; }
08213             return ret;
08214     });
08215 }
```

```

08214    };
08215
08216    if (res.has_header("Content-Length")) {
08217        if (!req.content_receiver) {
08218            auto len = res.get_header_value_u64("Content-Length");
08219            if (len > res.body.max_size()) {
08220                error = Error::Read;
08221                return false;
08222            }
08223            res.body.reserve(static_cast<size_t>(len));
08224        }
08225    }
08226
08227    if (res.status != StatusCode::NotModified_304) {
08228        int dummy_status;
08229        if (!detail::read_content(strm, res, (std::numeric_limits<size_t>::max)(),
08230            dummy_status, std::move(progress),
08231            std::move(out), decompress_)) {
08232            if (error != Error::Canceled) { error = Error::Read; }
08233            return false;
08234        }
08235    }
08236 }
08237
08238 // Log
08239 if (logger_) { logger_(req, res); }
08240
08241 return true;
08242 }
08243
08244 inline ContentProviderWithoutLength ClientImpl::get_multipart_content_provider(
08245     const std::string &boundary, const MultipartFormDataItems &items,
08246     const MultipartFormDataProviderItems &provider_items) const {
08247     size_t cur_item = 0;
08248     size_t cur_start = 0;
08249     // cur_item and cur_start are copied to within the std::function and maintain
08250     // state between successive calls
08251     return [&, cur_item, cur_start](size_t offset,
08252         DataSink &sink) mutable -> bool {
08253         if (!offset && !items.empty()) {
08254             sink.os << detail::serialize_multipart_formdata(items, boundary, false);
08255             return true;
08256         } else if (cur_item < provider_items.size()) {
08257             if (!cur_start) {
08258                 const auto &begin = detail::serialize_multipart_formdata_item_begin(
08259                     provider_items[cur_item], boundary);
08260                 offset += begin.size();
08261                 cur_start = offset;
08262                 sink.os << begin;
08263             }
08264
08265             DataSink cur_sink;
08266             auto has_data = true;
08267             cur_sink.write = sink.write;
08268             cur_sink.done = [&]() { has_data = false; };
08269
08270             if (!provider_items[cur_item].provider(offset - cur_start, cur_sink)) {
08271                 return false;
08272             }
08273
08274             if (!has_data) {
08275                 sink.os << detail::serialize_multipart_formdata_item_end();
08276                 cur_item++;
08277                 cur_start = 0;
08278             }
08279             return true;
08280         } else {
08281             sink.os << detail::serialize_multipart_formdata_finish(boundary);
08282             sink.done();
08283             return true;
08284         }
08285     };
08286 }
08287
08288 inline bool ClientImpl::process_socket(
08289     const Socket &socket,
08290     std::chrono::time_point<std::chrono::steady_clock> start_time,
08291     std::function<bool(Stream &strm)> callback) {
08292     return detail::process_client_socket(
08293         socket.sock, read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
08294         write_timeout_usec_, max_timeout_msec_, start_time, std::move(callback));
08295 }
08296
08297 inline bool ClientImpl::is_ssl() const { return false; }
08298
08299 inline Result ClientImpl::Get(const std::string &path) {
08300     return Get(path, Headers(), Progress());

```

```
08301 }
08302
08303 inline Result ClientImpl::Get(const std::string &path, Progress progress) {
08304     return Get(path, Headers(), std::move(progress));
08305 }
08306
08307 inline Result ClientImpl::Get(const std::string &path, const Headers &headers) {
08308     return Get(path, headers, Progress());
08309 }
08310
08311 inline Result ClientImpl::Get(const std::string &path, const Headers &headers,
08312             Progress progress) {
08313     Request req;
08314     req.method = "GET";
08315     req.path = path;
08316     req.headers = headers;
08317     req.progress = std::move(progress);
08318     if (max_timeout_msec_ > 0) {
08319         req.start_time_ = std::chrono::steady_clock::now();
08320     }
08321
08322     return send_(std::move(req));
08323 }
08324
08325 inline Result ClientImpl::Get(const std::string &path,
08326             ContentReceiver content_receiver) {
08327     return Get(path, Headers(), nullptr, std::move(content_receiver), nullptr);
08328 }
08329
08330 inline Result ClientImpl::Get(const std::string &path,
08331             ContentReceiver content_receiver,
08332             Progress progress) {
08333     return Get(path, Headers(), nullptr, std::move(content_receiver),
08334             std::move(progress));
08335 }
08336
08337 inline Result ClientImpl::Get(const std::string &path, const Headers &headers,
08338             ContentReceiver content_receiver) {
08339     return Get(path, headers, nullptr, std::move(content_receiver), nullptr);
08340 }
08341
08342 inline Result ClientImpl::Get(const std::string &path, const Headers &headers,
08343             ContentReceiver content_receiver,
08344             Progress progress) {
08345     return Get(path, headers, nullptr, std::move(content_receiver),
08346             std::move(progress));
08347 }
08348
08349 inline Result ClientImpl::Get(const std::string &path,
08350             ResponseHandler response_handler,
08351             ContentReceiver content_receiver) {
08352     return Get(path, Headers(), std::move(response_handler),
08353             std::move(content_receiver), nullptr);
08354 }
08355
08356 inline Result ClientImpl::Get(const std::string &path, const Headers &headers,
08357             ResponseHandler response_handler,
08358             ContentReceiver content_receiver) {
08359     return Get(path, headers, std::move(response_handler),
08360             std::move(content_receiver), nullptr);
08361 }
08362
08363 inline Result ClientImpl::Get(const std::string &path,
08364             ResponseHandler response_handler,
08365             ContentReceiver content_receiver,
08366             Progress progress) {
08367     return Get(path, Headers(), std::move(response_handler),
08368             std::move(content_receiver), std::move(progress));
08369 }
08370
08371 inline Result ClientImpl::Get(const std::string &path, const Headers &headers,
08372             ResponseHandler response_handler,
08373             ContentReceiver content_receiver,
08374             Progress progress) {
08375     Request req;
08376     req.method = "GET";
08377     req.path = path;
08378     req.headers = headers;
08379     req.response_handler = std::move(response_handler);
08380     req.content_receiver =
08381         [content_receiver](const char *data, size_t data_length,
08382                             uint64_t /*offset*/, uint64_t /*total_length*/) {
08383             return content_receiver(data, data_length);
08384         };
08385     req.progress = std::move(progress);
08386     if (max_timeout_msec_ > 0) {
08387         req.start_time_ = std::chrono::steady_clock::now();
```

```
08388  }
08389
08390  return send_(std::move(req));
08391 }
08392
08393 inline Result ClientImpl::Get(const std::string &path, const Params &params,
08394             const Headers &headers, Progress progress) {
08395  if (params.empty()) { return Get(path, headers); }
08396
08397  std::string path_with_query = append_query_params(path, params);
08398  return Get(path_with_query, headers, std::move(progress));
08399 }
08400
08401 inline Result ClientImpl::Get(const std::string &path, const Params &params,
08402             const Headers &headers,
08403             ContentReceiver content_receiver,
08404             Progress progress) {
08405  return Get(path, params, headers, nullptr, std::move(content_receiver),
08406             std::move(progress));
08407 }
08408
08409 inline Result ClientImpl::Get(const std::string &path, const Params &params,
08410             const Headers &headers,
08411             ResponseHandler response_handler,
08412             ContentReceiver content_receiver,
08413             Progress progress) {
08414  if (params.empty()) {
08415    return Get(path, headers, std::move(response_handler),
08416               std::move(content_receiver), std::move(progress));
08417 }
08418
08419  std::string path_with_query = append_query_params(path, params);
08420  return Get(path_with_query, headers, std::move(response_handler),
08421             std::move(content_receiver), std::move(progress));
08422 }
08423
08424 inline Result ClientImpl::Head(const std::string &path) {
08425  return Head(path, Headers());
08426 }
08427
08428 inline Result ClientImpl::Head(const std::string &path,
08429             const Headers &headers) {
08430  Request req;
08431  req.method = "HEAD";
08432  req.headers = headers;
08433  req.path = path;
08434  if (max_timeout_msec_ > 0) {
08435    req.start_time_ = std::chrono::steady_clock::now();
08436  }
08437
08438  return send_(std::move(req));
08439 }
08440
08441 inline Result ClientImpl::Post(const std::string &path) {
08442  return Post(path, std::string(), std::string());
08443 }
08444
08445 inline Result ClientImpl::Post(const std::string &path,
08446             const Headers &headers) {
08447  return Post(path, headers, nullptr, 0, std::string());
08448 }
08449
08450 inline Result ClientImpl::Post(const std::string &path, const char *body,
08451             size_t content_length,
08452             const std::string &content_type) {
08453  return Post(path, Headers(), body, content_length, content_type, nullptr);
08454 }
08455
08456 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08457             const char *body, size_t content_length,
08458             const std::string &content_type) {
08459  return send_with_content_provider("POST", path, headers, body, content_length,
08460             nullptr, nullptr, content_type, nullptr);
08461 }
08462
08463 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08464             const char *body, size_t content_length,
08465             const std::string &content_type,
08466             Progress progress) {
08467  return send_with_content_provider("POST", path, headers, body, content_length,
08468             nullptr, nullptr, content_type, progress);
08469 }
08470
08471 inline Result ClientImpl::Post(const std::string &path, const std::string &body,
08472             const std::string &content_type) {
08473  return Post(path, Headers(), body, content_type);
08474 }
```

```
08475
08476 inline Result ClientImpl::Post(const std::string &path, const std::string &body,
08477                                     const std::string &content_type,
08478                                     Progress progress) {
08479     return Post(path, Headers(), body, content_type, progress);
08480 }
08481
08482 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08483                                     const std::string &body,
08484                                     const std::string &content_type) {
08485     return send_with_content_provider("POST", path, headers, body.data(),
08486                                         body.size(), nullptr, nullptr, content_type,
08487                                         nullptr);
08488 }
08489
08490 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08491                                     const std::string &body,
08492                                     const std::string &content_type,
08493                                     Progress progress) {
08494     return send_with_content_provider("POST", path, headers, body.data(),
08495                                         body.size(), nullptr, nullptr, content_type,
08496                                         progress);
08497 }
08498
08499 inline Result ClientImpl::Post(const std::string &path, const Params &params) {
08500     return Post(path, Headers(), params);
08501 }
08502
08503 inline Result ClientImpl::Post(const std::string &path, size_t content_length,
08504                                     ContentProvider content_provider,
08505                                     const std::string &content_type) {
08506     return Post(path, Headers(), content_length, std::move(content_provider),
08507                 content_type);
08508 }
08509
08510 inline Result ClientImpl::Post(const std::string &path,
08511                                     ContentProviderWithoutLength content_provider,
08512                                     const std::string &content_type) {
08513     return Post(path, Headers(), std::move(content_provider), content_type);
08514 }
08515
08516 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08517                                     size_t content_length,
08518                                     ContentProvider content_provider,
08519                                     const std::string &content_type) {
08520     return send_with_content_provider("POST", path, headers, nullptr,
08521                                         content_length, std::move(content_provider),
08522                                         nullptr, content_type, nullptr);
08523 }
08524
08525 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08526                                     ContentProviderWithoutLength content_provider,
08527                                     const std::string &content_type) {
08528     return send_with_content_provider("POST", path, headers, nullptr, 0, nullptr,
08529                                         std::move(content_provider), content_type,
08530                                         nullptr);
08531 }
08532
08533 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08534                                     const Params &params) {
08535     auto query = detail::params_to_query_str(params);
08536     return Post(path, headers, query, "application/x-www-form-urlencoded");
08537 }
08538
08539 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08540                                     const Params &params, Progress progress) {
08541     auto query = detail::params_to_query_str(params);
08542     return Post(path, headers, query, "application/x-www-form-urlencoded",
08543                 progress);
08544 }
08545
08546 inline Result ClientImpl::Post(const std::string &path,
08547                                     const MultipartFormDataItems &items) {
08548     return Post(path, Headers(), items);
08549 }
08550
08551 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08552                                     const MultipartFormDataItems &items) {
08553     const auto &boundary = detail::make_multipart_data_boundary();
08554     const auto &content_type =
08555         detail::serialize_multipart_formdata_get_content_type(boundary);
08556     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08557     return Post(path, headers, body, content_type);
08558 }
08559
08560 inline Result ClientImpl::Post(const std::string &path, const Headers &headers,
08561                                     const MultipartFormDataItems &items,
```

```
08562             const std::string &boundary) {
08563     if (!detail::is_multipart_boundary_chars_valid(boundary)) {
08564         return Result{nullptr, Error::UnsupportedMultipartBoundaryChars};
08565     }
08566
08567     const auto &content_type =
08568         detail::serialize_multipart_formdata_get_content_type(boundary);
08569     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08570     return Post(path, headers, body, content_type);
08571 }
08572
08573 inline Result
08574 ClientImpl::Post(const std::string &path, const Headers &headers,
08575                     const MultipartFormDataItems &items,
08576                     const MultipartFormDataProviderItems &provider_items) {
08577     const auto &boundary = detail::make_multipart_data_boundary();
08578     const auto &content_type =
08579         detail::serialize_multipart_formdata_get_content_type(boundary);
08580     return send_with_content_provider(
08581         "POST", path, headers, nullptr, 0, nullptr,
08582         get_multipart_content_provider(boundary, items, provider_items),
08583         content_type, nullptr);
08584 }
08585
08586 inline Result ClientImpl::Put(const std::string &path) {
08587     return Put(path, std::string(), std::string());
08588 }
08589
08590 inline Result ClientImpl::Put(const std::string &path, const char *body,
08591                               size_t content_length,
08592                               const std::string &content_type) {
08593     return Put(path, Headers(), body, content_length, content_type);
08594 }
08595
08596 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08597                                 const char *body, size_t content_length,
08598                                 const std::string &content_type) {
08599     return send_with_content_provider("PUT", path, headers, body, content_length,
08600                                         nullptr, nullptr, content_type, nullptr);
08601 }
08602
08603 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08604                                 const char *body, size_t content_length,
08605                                 const std::string &content_type,
08606                                 Progress progress) {
08607     return send_with_content_provider("PUT", path, headers, body, content_length,
08608                                         nullptr, nullptr, content_type, progress);
08609 }
08610
08611 inline Result ClientImpl::Put(const std::string &path, const std::string &body,
08612                               const std::string &content_type) {
08613     return Put(path, Headers(), body, content_type);
08614 }
08615
08616 inline Result ClientImpl::Put(const std::string &path, const std::string &body,
08617                               const std::string &content_type,
08618                               Progress progress) {
08619     return Put(path, Headers(), body, content_type, progress);
08620 }
08621
08622 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08623                               const std::string &body,
08624                               const std::string &content_type) {
08625     return send_with_content_provider("PUT", path, headers, body.data(),
08626                                         body.size(), nullptr, nullptr, content_type,
08627                                         nullptr);
08628 }
08629
08630 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08631                               const std::string &body,
08632                               const std::string &content_type,
08633                               Progress progress) {
08634     return send_with_content_provider("PUT", path, headers, body.data(),
08635                                         body.size(), nullptr, nullptr, content_type,
08636                                         progress);
08637 }
08638
08639 inline Result ClientImpl::Put(const std::string &path, size_t content_length,
08640                               ContentProvider content_provider,
08641                               const std::string &content_type) {
08642     return Put(path, Headers(), content_length, std::move(content_provider),
08643                content_type);
08644 }
08645
08646 inline Result ClientImpl::Put(const std::string &path,
08647                               ContentProviderWithoutLength content_provider,
08648                               const std::string &content_type) {
```

```
08649 return Put(path, Headers(), std::move(content_provider), content_type);
08650 }
08651
08652 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08653     size_t content_length,
08654     ContentProvider content_provider,
08655     const std::string &content_type) {
08656     return send_with_content_provider("PUT", path, headers, nullptr,
08657         content_length, std::move(content_provider),
08658         nullptr, content_type, nullptr);
08659 }
08660
08661 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08662     ContentProviderWithoutLength content_provider,
08663     const std::string &content_type) {
08664     return send_with_content_provider("PUT", path, headers, nullptr, 0, nullptr,
08665         std::move(content_provider), content_type,
08666         nullptr);
08667 }
08668
08669 inline Result ClientImpl::Put(const std::string &path, const Params &params) {
08670     return Put(path, Headers(), params);
08671 }
08672
08673 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08674     const Params &params) {
08675     auto query = detail::params_to_query_str(params);
08676     return Put(path, headers, query, "application/x-www-form-urlencoded");
08677 }
08678
08679 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08680     const Params &params, Progress progress) {
08681     auto query = detail::params_to_query_str(params);
08682     return Put(path, headers, query, "application/x-www-form-urlencoded",
08683         progress);
08684 }
08685
08686 inline Result ClientImpl::Put(const std::string &path,
08687     const MultipartFormDataItems &items) {
08688     return Put(path, Headers(), items);
08689 }
08690
08691 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08692     const MultipartFormDataItems &items) {
08693     const auto &boundary = detail::make_multipart_data_boundary();
08694     const auto &content_type =
08695         detail::serialize_multipart_formdata_get_content_type(boundary);
08696     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08697     return Put(path, headers, body, content_type);
08698 }
08699
08700 inline Result ClientImpl::Put(const std::string &path, const Headers &headers,
08701     const MultipartFormDataItems &items,
08702     const std::string &boundary) {
08703     if (!detail::is_multipart_boundary_chars_valid(boundary)) {
08704         return Result{nullptr, Error::UnsupportedMultipartBoundaryChars};
08705     }
08706
08707     const auto &content_type =
08708         detail::serialize_multipart_formdata_get_content_type(boundary);
08709     const auto &body = detail::serialize_multipart_formdata(items, boundary);
08710     return Put(path, headers, body, content_type);
08711 }
08712
08713 inline Result
08714 ClientImpl::Put(const std::string &path, const Headers &headers,
08715     const MultipartFormDataItems &items,
08716     const MultipartFormDataProviderItems &provider_items) {
08717     const auto &boundary = detail::make_multipart_data_boundary();
08718     const auto &content_type =
08719         detail::serialize_multipart_formdata_get_content_type(boundary);
08720     return send_with_content_provider(
08721         "PUT", path, headers, nullptr, 0, nullptr,
08722         get_multipart_content_provider(boundary, items, provider_items),
08723         content_type, nullptr);
08724 }
08725 inline Result ClientImpl::Patch(const std::string &path) {
08726     return Patch(path, std::string(), std::string());
08727 }
08728
08729 inline Result ClientImpl::Patch(const std::string &path, const char *body,
08730     size_t content_length,
08731     const std::string &content_type) {
08732     return Patch(path, Headers(), body, content_length, content_type);
08733 }
08734
08735 inline Result ClientImpl::Patch(const std::string &path, const char *body,
```

```
08736             size_t content_length,
08737             const std::string &content_type,
08738             Progress progress) {
08739     return Patch(path, Headers(), body, content_length, content_type, progress);
08740 }
08741
08742 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08743         const char *body, size_t content_length,
08744         const std::string &content_type) {
08745     return Patch(path, headers, body, content_length, content_type, nullptr);
08746 }
08747
08748 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08749         const char *body, size_t content_length,
08750         const std::string &content_type,
08751         Progress progress) {
08752     return send_with_content_provider("PATCH", path, headers, body,
08753             content_length, nullptr, nullptr,
08754             content_type, progress);
08755 }
08756
08757 inline Result ClientImpl::Patch(const std::string &path,
08758         const std::string &body,
08759         const std::string &content_type) {
08760     return Patch(path, Headers(), body, content_type);
08761 }
08762
08763 inline Result ClientImpl::Patch(const std::string &path,
08764         const std::string &body,
08765         const std::string &content_type,
08766         Progress progress) {
08767     return Patch(path, Headers(), body, content_type, progress);
08768 }
08769
08770 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08771         const std::string &body,
08772         const std::string &content_type) {
08773     return Patch(path, headers, body, content_type, nullptr);
08774 }
08775
08776 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08777         const std::string &body,
08778         const std::string &content_type,
08779         Progress progress) {
08780     return send_with_content_provider("PATCH", path, headers, body.data(),
08781             body.size(), nullptr, nullptr, content_type,
08782             progress);
08783 }
08784
08785 inline Result ClientImpl::Patch(const std::string &path, size_t content_length,
08786         ContentProvider content_provider,
08787         const std::string &content_type) {
08788     return Patch(path, Headers(), content_length, std::move(content_provider),
08789             content_type);
08790 }
08791
08792 inline Result ClientImpl::Patch(const std::string &path,
08793         ContentProviderWithoutLength content_provider,
08794         const std::string &content_type) {
08795     return Patch(path, Headers(), std::move(content_provider), content_type);
08796 }
08797
08798 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08799         size_t content_length,
08800         ContentProvider content_provider,
08801         const std::string &content_type) {
08802     return send_with_content_provider("PATCH", path, headers, nullptr,
08803             content_length, std::move(content_provider),
08804             nullptr, content_type, nullptr);
08805 }
08806
08807 inline Result ClientImpl::Patch(const std::string &path, const Headers &headers,
08808         ContentProviderWithoutLength content_provider,
08809         const std::string &content_type) {
08810     return send_with_content_provider("PATCH", path, headers, nullptr, 0, nullptr,
08811             std::move(content_provider), content_type,
08812             nullptr);
08813 }
08814
08815 inline Result ClientImpl::Delete(const std::string &path) {
08816     return Delete(path, Headers(), std::string(), std::string());
08817 }
08818
08819 inline Result ClientImpl::Delete(const std::string &path,
08820         const Headers &headers) {
08821     return Delete(path, headers, std::string(), std::string());
08822 }
```

```
08823
08824 inline Result ClientImpl::Delete(const std::string &path, const char *body,
08825                     size_t content_length,
08826                     const std::string &content_type) {
08827     return Delete(path, Headers(), body, content_length, content_type);
08828 }
08829
08830 inline Result ClientImpl::Delete(const std::string &path, const char *body,
08831                     size_t content_length,
08832                     const std::string &content_type,
08833                     Progress progress) {
08834     return Delete(path, Headers(), body, content_length, content_type, progress);
08835 }
08836
08837 inline Result ClientImpl::Delete(const std::string &path,
08838                     const Headers &headers, const char *body,
08839                     size_t content_length,
08840                     const std::string &content_type) {
08841     return Delete(path, headers, body, content_length, content_type, nullptr);
08842 }
08843
08844 inline Result ClientImpl::Delete(const std::string &path,
08845                     const Headers &headers, const char *body,
08846                     size_t content_length,
08847                     const std::string &content_type,
08848                     Progress progress) {
08849     Request req;
08850     req.method = "DELETE";
08851     req.headers = headers;
08852     req.path = path;
08853     req.progress = progress;
08854     if (max_timeout_msec_ > 0) {
08855         req.start_time_ = std::chrono::steady_clock::now();
08856     }
08857
08858     if (!content_type.empty()) { req.set_header("Content-Type", content_type); }
08859     req.body.assign(body, content_length);
08860
08861     return send_(std::move(req));
08862 }
08863
08864 inline Result ClientImpl::Delete(const std::string &path,
08865                     const std::string &body,
08866                     const std::string &content_type) {
08867     return Delete(path, Headers(), body.data(), body.size(), content_type);
08868 }
08869
08870 inline Result ClientImpl::Delete(const std::string &path,
08871                     const std::string &body,
08872                     const std::string &content_type,
08873                     Progress progress) {
08874     return Delete(path, Headers(), body.data(), body.size(), content_type,
08875                 progress);
08876 }
08877
08878 inline Result ClientImpl::Delete(const std::string &path,
08879                     const Headers &headers,
08880                     const std::string &body,
08881                     const std::string &content_type) {
08882     return Delete(path, headers, body.data(), body.size(), content_type);
08883 }
08884
08885 inline Result ClientImpl::Delete(const std::string &path,
08886                     const Headers &headers,
08887                     const std::string &body,
08888                     const std::string &content_type,
08889                     Progress progress) {
08890     return Delete(path, headers, body.data(), body.size(), content_type,
08891                 progress);
08892 }
08893
08894 inline Result ClientImpl::Options(const std::string &path) {
08895     return Options(path, Headers());
08896 }
08897
08898 inline Result ClientImpl::Options(const std::string &path,
08899                     const Headers &headers) {
08900     Request req;
08901     req.method = "OPTIONS";
08902     req.headers = headers;
08903     req.path = path;
08904     if (max_timeout_msec_ > 0) {
08905         req.start_time_ = std::chrono::steady_clock::now();
08906     }
08907
08908     return send_(std::move(req));
08909 }
```

```
08910
08911 inline void ClientImpl::stop() {
08912     std::lock_guard<std::mutex> guard(socket_mutex_);
08913
08914     // If there is anything ongoing right now, the ONLY thread-safe thing we can
08915     // do is shutdown_socket, so that threads using this socket suddenly
08916     // discover they can't read/write any more and error out. Everything else
08917     // (closing the socket, shutting ssl down) is unsafe because these actions are
08918     // not thread-safe.
08919     if (socket_requests_in_flight_ > 0) {
08920         shutdown_socket(socket_);
08921
08922     // Aside from that, we set a flag for the socket to be closed when we're
08923     // done.
08924     socket_should_be_closed_when_request_is_done_ = true;
08925     return;
08926 }
08927
08928 // Otherwise, still holding the mutex, we can shut everything down ourselves
08929 shutdown_ssl(socket_, true);
08930 shutdown_socket(socket_);
08931 close_socket(socket_);
08932 }
08933
08934 inline std::string ClientImpl::host() const { return host_; }
08935
08936 inline int ClientImpl::port() const { return port_; }
08937
08938 inline size_t ClientImpl::is_socket_open() const {
08939     std::lock_guard<std::mutex> guard(socket_mutex_);
08940     return socket_.is_open();
08941 }
08942
08943 inline socket_t ClientImpl::socket() const { return socket_.sock; }
08944
08945 inline void ClientImpl::set_connection_timeout(time_t sec, time_t usec) {
08946     connection_timeout_sec_ = sec;
08947     connection_timeout_usec_ = usec;
08948 }
08949
08950 inline void ClientImpl::set_read_timeout(time_t sec, time_t usec) {
08951     read_timeout_sec_ = sec;
08952     read_timeout_usec_ = usec;
08953 }
08954
08955 inline void ClientImpl::set_write_timeout(time_t sec, time_t usec) {
08956     write_timeout_sec_ = sec;
08957     write_timeout_usec_ = usec;
08958 }
08959
08960 inline void ClientImpl::set_max_timeout(time_t msec) {
08961     max_timeout_msec_ = msec;
08962 }
08963
08964 inline void ClientImpl::set_basic_auth(const std::string &username,
08965                                     const std::string &password) {
08966     basic_auth_username_ = username;
08967     basic_auth_password_ = password;
08968 }
08969
08970 inline void ClientImpl::set_bearer_token_auth(const std::string &token) {
08971     bearer_token_auth_token_ = token;
08972 }
08973
08974 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
08975 inline void ClientImpl::set_digest_auth(const std::string &username,
08976                                         const std::string &password) {
08977     digest_auth_username_ = username;
08978     digest_auth_password_ = password;
08979 }
08980 #endif
08981
08982 inline void ClientImpl::set_keep_alive(bool on) { keep_alive_ = on; }
08983
08984 inline void ClientImpl::set_follow_location(bool on) { follow_location_ = on; }
08985
08986 inline void ClientImpl::set_url_encode(bool on) { url_encode_ = on; }
08987
08988 inline void
08989 ClientImpl::set_hostname_addr_map(std::map<std::string, std::string> addr_map) {
08990     addr_map_ = std::move(addr_map);
08991 }
08992
08993 inline void ClientImpl::set_default_headers(Headers headers) {
08994     default_headers_ = std::move(headers);
08995 }
08996
```

```
08997 inline void ClientImpl::set_header_writer(
08998     std::function<ssize_t(Stream &, Headers &)>> const &writer) {
08999     header_writer_ = writer;
09000 }
09001
09002 inline void ClientImpl::set_address_family(int family) {
09003     address_family_ = family;
09004 }
09005
09006 inline void ClientImpl::set_tcp_nodelay(bool on) { tcp_nodelay_ = on; }
09007
09008 inline void ClientImpl::set_ipv6_v6only(bool on) { ipv6_v6only_ = on; }
09009
09010 inline void ClientImpl::set_socket_options(SocketOptions socket_options) {
09011     socket_options_ = std::move(socket_options);
09012 }
09013
09014 inline void ClientImpl::set_compress(bool on) { compress_ = on; }
09015
09016 inline void ClientImpl::set_decompress(bool on) { decompress_ = on; }
09017
09018 inline void ClientImpl::set_interface(const std::string &intf) {
09019     interface_ = intf;
09020 }
09021
09022 inline void ClientImpl::set_proxy(const std::string &host, int port) {
09023     proxy_host_ = host;
09024     proxy_port_ = port;
09025 }
09026
09027 inline void ClientImpl::set_proxy_basic_auth(const std::string &username,
09028                                                 const std::string &password) {
09029     proxy_basic_auth_username_ = username;
09030     proxy_basic_auth_password_ = password;
09031 }
09032
09033 inline void ClientImpl::set_proxy_bearer_token_auth(const std::string &token) {
09034     proxy_bearer_token_auth_token_ = token;
09035 }
09036
09037 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
09038 inline void ClientImpl::set_proxy_digest_auth(const std::string &username,
09039                                                 const std::string &password) {
09040     proxy_digest_auth_username_ = username;
09041     proxy_digest_auth_password_ = password;
09042 }
09043
09044 inline void ClientImpl::set_ca_cert_path(const std::string &ca_cert_file_path,
09045                                         const std::string &ca_cert_dir_path) {
09046     ca_cert_file_path_ = ca_cert_file_path;
09047     ca_cert_dir_path_ = ca_cert_dir_path;
09048 }
09049
09050 inline void ClientImpl::set_ca_cert_store(X509_STORE *ca_cert_store) {
09051     if (ca_cert_store && ca_cert_store != ca_cert_store_) {
09052         ca_cert_store_ = ca_cert_store;
09053     }
09054 }
09055
09056 inline X509_STORE *ClientImpl::create_ca_cert_store(const char *ca_cert,
09057                                                       std::size_t size) const {
09058     auto mem = BIO_new_mem_buf(ca_cert, static_cast<int>(size));
09059     auto se = detail::scope_exit([&] { BIO_free_all(mem); });
09060     if (!mem) { return nullptr; }
09061
09062     auto inf = PEM_X509_INFO_read_bio(mem, nullptr, nullptr, nullptr);
09063     if (!inf) { return nullptr; }
09064
09065     auto cts = X509_STORE_new();
09066     if (cts) {
09067         for (auto i = 0; i < static_cast<int>(sk_X509_INFO_num(inf)); i++) {
09068             auto itmp = sk_X509_INFO_value(inf, i);
09069             if (!itmp) { continue; }
09070
09071             if (itmp->x509) { X509_STORE_add_cert(cts, itmp->x509); }
09072             if (itmp->crl) { X509_STORE_add_crl(cts, itmp->crl); }
09073         }
09074     }
09075
09076     sk_X509_INFO_pop_free(inf, X509_INFO_free);
09077     return cts;
09078 }
09079
09080 inline void ClientImpl::enable_server_certificate_verification(bool enabled) {
09081     server_certificate_verification_ = enabled;
09082 }
09083
```

```

09084 inline void ClientImpl::enable_server_hostname_verification(bool enabled) {
09085     server_hostname_verification_ = enabled;
09086 }
09087
09088 inline void ClientImpl::set_server_certificate_verifier(
09089     std::function<SSLVerifierResponse(SSL *ssl)> verifier) {
09090     server_certificate_verifier_ = verifier;
09091 }
09092 #endif
09093
09094 inline void ClientImpl::set_logger(Logger logger) {
09095     logger_ = std::move(logger);
09096 }
09097
09098 /*
09099 * SSL Implementation
09100 */
09101 #ifndef CPPHTTPLIB_OPENSSL_SUPPORT
09102 namespace detail {
09103
09104 template <typename U, typename V>
09105 inline SSL *ssl_new(socket_t sock, SSL_CTX *ctx, std::mutex &ctx_mutex,
09106                      U SSL_connect_or_accept, V setup) {
09107     SSL *ssl = nullptr;
09108     {
09109         std::lock_guard<std::mutex> guard(ctx_mutex);
09110         ssl = SSL_new(ctx);
09111     }
09112
09113     if (ssl) {
09114         set_nonblocking(sock, true);
09115         auto bio = BIO_new_socket(static_cast<int>(sock), BIO_NOCLOSE);
09116         BIO_set_nbio(bio, 1);
09117         SSL_set_bio(ssl, bio, bio);
09118
09119         if (!setup(ssl) || SSL_connect_or_accept(ssl) != 1) {
09120             SSL_shutdown(ssl);
09121             {
09122                 std::lock_guard<std::mutex> guard(ctx_mutex);
09123                 SSL_free(ssl);
09124             }
09125             set_nonblocking(sock, false);
09126             return nullptr;
09127         }
09128         BIO_set_nbio(bio, 0);
09129         set_nonblocking(sock, false);
09130     }
09131
09132     return ssl;
09133 }
09134
09135 inline void ssl_delete(std::mutex &ctx_mutex, SSL *ssl, socket_t sock,
09136                         bool shutdown_gracefully) {
09137     // sometimes we may want to skip this to try to avoid SIGPIPE if we know
09138     // the remote has closed the network connection
09139     // Note that it is not always possible to avoid SIGPIPE, this is merely a
09140     // best-efforts.
09141     if (shutdown_gracefully) {
09142         (void)(sock);
09143         // SSL_shutdown() returns 0 on first call (indicating close_notify alert
09144         // sent) and 1 on subsequent call (indicating close_notify alert received)
09145         if (SSL_shutdown(ssl) == 0) {
09146             // Expected to return 1, but even if it doesn't, we free ssl
09147             SSL_shutdown(ssl);
09148         }
09149     }
09150
09151     std::lock_guard<std::mutex> guard(ctx_mutex);
09152     SSL_free(ssl);
09153 }
09154
09155 template <typename U>
09156 bool ssl_connect_or_accept_nonblocking(socket_t sock, SSL *ssl,
09157                                         U SSL_connect_or_accept,
09158                                         time_t timeout_sec,
09159                                         time_t timeout_usec) {
09160     auto res = 0;
09161     while ((res = SSL_connect_or_accept(ssl)) != 1) {
09162         auto err = SSL_get_error(ssl, res);
09163         switch (err) {
09164             case SSL_ERROR_WANT_READ:
09165                 if (select_read(sock, timeout_sec, timeout_usec) > 0) { continue; }
09166                 break;
09167             case SSL_ERROR_WANT_WRITE:
09168                 if (select_write(sock, timeout_sec, timeout_usec) > 0) { continue; }
09169                 break;
09170             default: break;

```

```

09171     }
09172     return false;
09173   }
09174   return true;
09175 }
09176
09177 template <typename T>
09178 inline bool process_server_socket_ssl(
09179   const std::atomic<socket_t> &svr_sock, SSL *ssl, socket_t sock,
09180   size_t keep_alive_max_count, time_t keep_alive_timeout_sec,
09181   time_t read_timeout_sec, time_t read_timeout_usec, time_t write_timeout_sec,
09182   time_t write_timeout_usec, T callback) {
09183   return process_server_socket_core(
09184     svr_sock, sock, keep_alive_max_count, keep_alive_timeout_sec,
09185     [&](bool close_connection, bool &connection_closed) {
09186       SSLSocketStream strm(sock, ssl, read_timeout_sec, read_timeout_usec,
09187                             write_timeout_sec, write_timeout_usec);
09188       return callback(strm, close_connection, connection_closed);
09189     });
09190 }
09191
09192 template <typename T>
09193 inline bool process_client_socket_ssl(
09194   SSL *ssl, socket_t sock, time_t read_timeout_sec, time_t read_timeout_usec,
09195   time_t write_timeout_sec, time_t write_timeout_usec,
09196   time_t max_timeout_msec,
09197   std::chrono::time_point<std::chrono::steady_clock> start_time, T callback) {
09198   SSLSocketStream strm(sock, ssl, read_timeout_sec, read_timeout_usec,
09199                         write_timeout_sec, write_timeout_usec, max_timeout_msec,
09200                         start_time);
09201   return callback(strm);
09202 }
09203
09204 // SSL socket stream implementation
09205 inline SSLSocketStream::SSLSocketStream(
09206   socket_t sock, SSL *ssl, time_t read_timeout_sec, time_t read_timeout_usec,
09207   time_t write_timeout_sec, time_t write_timeout_usec,
09208   time_t max_timeout_msec,
09209   std::chrono::time_point<std::chrono::steady_clock> start_time)
09210 : sock_(sock), ssl_(ssl), read_timeout_sec_(read_timeout_sec),
09211   read_timeout_usec_(read_timeout_usec),
09212   write_timeout_sec_(write_timeout_sec),
09213   write_timeout_usec_(write_timeout_usec),
09214   max_timeout_msec_(max_timeout_msec), start_time_(start_time) {
09215   SSL_clear_mode(ssl, SSL_MODE_AUTO_RETRY);
09216 }
09217
09218 inline SSLSocketStream::~SSLSocketStream() = default;
09219
09220 inline bool SSLSocketStream::is_readable() const {
09221   return SSL_pending(ssl_) > 0;
09222 }
09223
09224 inline bool SSLSocketStream::wait_readable() const {
09225   if (max_timeout_msec_ <= 0) {
09226     return select_read(sock_, read_timeout_sec_, read_timeout_usec_) > 0;
09227   }
09228
09229   time_t read_timeout_sec;
09230   time_t read_timeout_usec;
09231   calc_actual_timeout(max_timeout_msec_, duration(), read_timeout_sec_,
09232                       read_timeout_usec_, read_timeout_sec, read_timeout_usec);
09233
09234   return select_read(sock_, read_timeout_sec, read_timeout_usec) > 0;
09235 }
09236
09237 inline bool SSLSocketStream::wait_writable() const {
09238   return select_write(sock_, write_timeout_sec_, write_timeout_usec_) > 0 &&
09239     is_socket_alive(sock_) && !is_ssl_peer_could_be_closed(ssl_, sock_);
09240 }
09241
09242 inline ssize_t SSLSocketStream::read(char *ptr, size_t size) {
09243   if (SSL_pending(ssl_) > 0) {
09244     return SSL_read(ssl_, ptr, static_cast<int>(size));
09245   } else if (wait_readable()) {
09246     auto ret = SSL_read(ssl_, ptr, static_cast<int>(size));
09247     if (ret < 0) {
09248       auto err = SSL_get_error(ssl_, ret);
09249       auto n = 1000;
09250 #ifdef _WIN32
09251       while (--n >= 0 && (err == SSL_ERROR_WANT_READ ||
09252                               (err == SSL_ERROR_SYSCALL &&
09253                                 WSAGetLastError() == WSAETIMEDOUT))) {
09254 #else
09255       while (--n >= 0 && err == SSL_ERROR_WANT_READ) {
09256 #endif
09257       if (SSL_pending(ssl_) > 0) {

```

```

09258     return SSL_read(ssl_, ptr, static_cast<int>(size));
09259 } else if (wait_readable()) {
09260     std::this_thread::sleep_for(std::chrono::microseconds{10});
09261     ret = SSL_read(ssl_, ptr, static_cast<int>(size));
09262     if (ret >= 0) { return ret; }
09263     err = SSL_get_error(ssl_, ret);
09264 } else {
09265     return -1;
09266 }
09267 }
09268 }
09269 return ret;
09270 } else {
09271     return -1;
09272 }
09273 }
09274
09275 inline ssize_t SSLSocketStream::write(const char *ptr, size_t size) {
09276 if (wait_writable()) {
09277     auto handle_size = static_cast<int>(
09278         std::min<size_t>(size, (std::numeric_limits<int>::max)()));
09279
09280     auto ret = SSL_write(ssl_, ptr, static_cast<int>(handle_size));
09281     if (ret < 0) {
09282         auto err = SSL_get_error(ssl_, ret);
09283         auto n = 1000;
09284 #ifdef WIN32
09285         while (--n >= 0 && (err == SSL_ERROR_WANT_WRITE ||
09286             (err == SSL_ERROR_SYSCALL &&
09287                 WSAGetLastError() == WSAETIMEDOUT))) {
09288 #else
09289         while (--n >= 0 && err == SSL_ERROR_WANT_WRITE) {
09290 #endif
09291         if (wait_writable()) {
09292             std::this_thread::sleep_for(std::chrono::microseconds{10});
09293             ret = SSL_write(ssl_, ptr, static_cast<int>(handle_size));
09294             if (ret >= 0) { return ret; }
09295             err = SSL_get_error(ssl_, ret);
09296         } else {
09297             return -1;
09298         }
09299     }
09300 }
09301     return ret;
09302 }
09303 return -1;
09304 }
09305
09306 inline void SSLSocketStream::get_remote_ip_and_port(std::string &ip,
09307                                         int &port) const {
09308     detail::get_remote_ip_and_port(sock_, ip, port);
09309 }
09310
09311 inline void SSLSocketStream::get_local_ip_and_port(std::string &ip,
09312                                         int &port) const {
09313     detail::get_local_ip_and_port(sock_, ip, port);
09314 }
09315
09316 inline socket_t SSLSocketStream::socket() const { return sock_; }
09317
09318 inline time_t SSLSocketStream::duration() const {
09319     return std::chrono::duration_cast<std::chrono::milliseconds>(
09320         std::chrono::steady_clock::now() - start_time_)
09321         .count();
09322 }
09323
09324 } // namespace detail
09325
09326 // SSL HTTP server implementation
09327 inline SSLServer::SSLServer(const char *cert_path, const char *private_key_path,
09328                             const char *client_ca_cert_file_path,
09329                             const char *client_ca_cert_dir_path,
09330                             const char *private_key_password) {
09331     ctx_ = SSL_CTX_new(TLS_server_method());
09332
09333     if (ctx_) {
09334         SSL_CTX_set_options(ctx_,
09335             SSL_OP_NO_COMPRESSION |
09336             SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION);
09337
09338     SSL_CTX_set_min_proto_version(ctx_, TLS1_2_VERSION);
09339
09340     if (private_key_password != nullptr && (private_key_password[0] != '\0')) {
09341         SSL_CTX_set_default_passwd_cb_userdata(
09342             ctx_,
09343             reinterpret_cast<void *>(const_cast<char *>(private_key_password)));
09344     }

```

```

09345
09346 if (SSL_CTX_use_certificate_chain_file(ctx_, cert_path) != 1 ||
09347     SSL_CTX_use_PrivateKey_file(ctx_, private_key_path, SSL_FILETYPE_PEM) !=
09348     1 ||
09349     SSL_CTX_check_private_key(ctx_) != 1) {
09350     SSL_CTX_free(ctx_);
09351     ctx_ = nullptr;
09352 } else_if (client_ca_cert_file_path || client_ca_cert_dir_path) {
09353     SSL_CTX_load_verify_locations(ctx_, client_ca_cert_file_path,
09354         client_ca_cert_dir_path);
09355
09356     SSL_CTX_set_verify(
09357         ctx_, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, nullptr);
09358 }
09359 }
09360 }
09361
09362 inline SSLServer::SSLServer(X509 *cert, EVP_PKEY *private_key,
09363             X509_STORE *client_ca_cert_store) {
09364     ctx_ = SSL_CTX_new(TLS_server_method());
09365
09366     if (ctx_) {
09367         SSL_CTX_set_options(ctx_,
09368             SSL_OP_NO_COMPRESSION |
09369             SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION);
09370
09371     SSL_CTX_set_min_proto_version(ctx_, TLS1_2_VERSION);
09372
09373     if (SSL_CTX_use_certificate(ctx_, cert) != 1 ||
09374         SSL_CTX_use_PrivateKey(ctx_, private_key) != 1) {
09375         SSL_CTX_free(ctx_);
09376         ctx_ = nullptr;
09377     } else_if (client_ca_cert_store) {
09378         SSL_CTX_set_cert_store(ctx_, client_ca_cert_store);
09379
09380         SSL_CTX_set_verify(
09381             ctx_, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, nullptr);
09382     }
09383 }
09384 }
09385
09386 inline SSLServer::SSLServer(
09387     const std::function<bool(SSL_CTX &ssl_ctx)> &setup_ssl_ctx_callback) {
09388     ctx_ = SSL_CTX_new(TLS_method());
09389     if (ctx_) {
09390         if (!setup_ssl_ctx_callback(*ctx_)) {
09391             SSL_CTX_free(ctx_);
09392             ctx_ = nullptr;
09393         }
09394     }
09395 }
09396
09397 inline SSLServer::~SSLServer() {
09398     if (ctx_) { SSL_CTX_free(ctx_); }
09399 }
09400
09401 inline bool SSLServer::is_valid() const { return ctx_; }
09402
09403 inline SSL_CTX *SSLServer::ssl_context() const { return ctx_; }
09404
09405 inline void SSLServer::update_certs(X509 *cert, EVP_PKEY *private_key,
09406             X509_STORE *client_ca_cert_store) {
09407
09408     std::lock_guard<std::mutex> guard(ctx_mutex_);
09409
09410     SSL_CTX_use_certificate(ctx_, cert);
09411     SSL_CTX_use_PrivateKey(ctx_, private_key);
09412
09413     if (client_ca_cert_store != nullptr) {
09414         SSL_CTX_set_cert_store(ctx_, client_ca_cert_store);
09415     }
09416 }
09417
09418 inline bool SSLServer::process_and_close_socket(socket_t sock) {
09419     auto ssl = detail::ssl_new(
09420         sock, ctx_, ctx_mutex_,
09421         [&](SSL *ssl2) {
09422             return detail::ssl_connect_or_accept_nonblocking(
09423                 sock, ssl2, SSL_ACCEPT, read_timeout_sec_, read_timeout_usec_);
09424         },
09425         [(SSL * /*ssl2*/)] { return true; });
09426
09427     auto ret = false;
09428     if (ssl) {
09429         std::string remote_addr;
09430         int remote_port = 0;
09431         detail::get_remote_ip_and_port(sock, remote_addr, remote_port);

```

```

09432
09433     std::string local_addr;
09434     int local_port = 0;
09435     detail::get_local_ip_and_port(sock, local_addr, local_port);
09436
09437     ret = detail::process_server_socket_ssl(
09438         svr_sock_, ssl, sock, keep_alive_max_count_, keep_alive_timeout_sec_,
09439         read_timeout_sec_, read_timeout_usec_, write_timeout_sec_,
09440         write_timeout_usec_,
09441         [&](Stream &strm, bool close_connection, bool &connection_closed) {
09442             return process_request(strm, remote_addr, remote_port, local_addr,
09443                                     local_port, close_connection,
09444                                     connection_closed,
09445                                     [&](Request &req) { req.ssl = ssl; });
09446         });
09447
09448     // Shutdown gracefully if the result seemed successful, non-gracefully if
09449     // the connection appeared to be closed.
09450     const bool shutdown_gracefully = ret;
09451     detail::ssl_delete(ctx_mutex_, ssl, sock, shutdown_gracefully);
09452 }
09453
09454     detail::shutdown_socket(sock);
09455     detail::close_socket(sock);
09456     return ret;
09457 }
09458
09459 // SSL HTTP client implementation
09460 inline SSLClient::SSLClient(const std::string &host)
09461     : SSLClient(host, 443, std::string(), std::string()) {}
09462
09463 inline SSLClient::SSLClient(const std::string &host, int port)
09464     : SSLClient(host, port, std::string(), std::string()) {}
09465
09466 inline SSLClient::SSLClient(const std::string &host, int port,
09467     const std::string &client_cert_path,
09468     const std::string &client_key_path,
09469     const std::string &private_key_password)
09470     : ClientImpl(host, port, client_cert_path, client_key_path) {
09471     ctx_ = SSL_CTX_new(TLS_client_method());
09472
09473     SSL_CTX_set_min_proto_version(ctx_, TLS1_2_VERSION);
09474
09475     detail::split(&host_[0], &host_[host_.size()], ',',
09476                 [&](const char *b, const char *e) {
09477                     host_components_.emplace_back(b, e);
09478                 });
09479
09480     if (!client_cert_path.empty() && !client_key_path.empty()) {
09481         if (!private_key_password.empty()) {
09482             SSL_CTX_set_default_passwd_cb_userdata(
09483                 ctx_, reinterpret_cast<void*>(
09484                     const_cast<char*>(private_key_password.c_str())));
09485         }
09486
09487         if (SSL_CTX_use_certificate_file(ctx_, client_cert_path.c_str(),
09488                                         SSL_FILETYPE_PEM) != -1 ||
09489             SSL_CTX_use_PrivateKey_file(ctx_, client_key_path.c_str(),
09490                                         SSL_FILETYPE_PEM) != -1) {
09491             SSL_CTX_free(ctx_);
09492             ctx_ = nullptr;
09493         }
09494     }
09495 }
09496
09497 inline SSLClient::SSLClient(const std::string &host, int port,
09498     X509 *client_cert, EVP_PKEY *client_key,
09499     const std::string &private_key_password)
09500     : ClientImpl(host, port) {
09501     ctx_ = SSL_CTX_new(TLS_client_method());
09502
09503     detail::split(&host_[0], &host_[host_.size()], ',',
09504                 [&](const char *b, const char *e) {
09505                     host_components_.emplace_back(b, e);
09506                 });
09507
09508     if (client_cert != nullptr && client_key != nullptr) {
09509         if (!private_key_password.empty()) {
09510             SSL_CTX_set_default_passwd_cb_userdata(
09511                 ctx_, reinterpret_cast<void*>(
09512                     const_cast<char*>(private_key_password.c_str())));
09513         }
09514
09515         if (SSL_CTX_use_certificate(ctx_, client_cert) != 1 ||
09516             SSL_CTX_use_PrivateKey(ctx_, client_key) != 1) {
09517             SSL_CTX_free(ctx_);
09518             ctx_ = nullptr;

```

```

09519     }
09520 }
09521 }
09522
09523 inline SSLClient::~SSLClient() {
09524     if (ctx_) { SSL_CTX_free(ctx_); }
09525     // Make sure to shut down SSL since shutdown_ssl will resolve to the
09526     // base function rather than the derived function once we get to the
09527     // base class destructor, and won't free the SSL (causing a leak).
09528     shutdown_sslImpl(socket_, true);
09529 }
09530
09531 inline bool SSLClient::is_valid() const { return ctx_; }
09532
09533 inline void SSLClient::set_ca_cert_store(X509_STORE *ca_cert_store) {
09534     if (ca_cert_store) {
09535         if (ctx_) {
09536             if (SSL_CTX_get_cert_store(ctx_) != ca_cert_store) {
09537                 // Free memory allocated for old cert and use new store `ca_cert_store`
09538                 SSL_CTX_set_cert_store(ctx_, ca_cert_store);
09539             }
09540         } else {
09541             X509_STORE_free(ca_cert_store);
09542         }
09543     }
09544 }
09545
09546 inline void SSLClient::load_ca_cert_store(const char *ca_cert,
09547                                         std::size_t size) {
09548     set_ca_cert_store(ClientImpl::create_ca_cert_store(ca_cert, size));
09549 }
09550
09551 inline long SSLClient::get_openssl_verify_result() const {
09552     return verify_result_;
09553 }
09554
09555 inline SSL_CTX *SSLClient::ssl_context() const { return ctx_; }
09556
09557 inline bool SSLClient::create_and_connect_socket(Socket &socket, Error &error) {
09558     return is_valid() && ClientImpl::create_and_connect_socket(socket, error);
09559 }
09560
09561 // Assumes that socket_mutex_ is locked and that there are no requests in flight
09562 inline bool SSLClient::connect_with_proxy(
09563     Socket &socket,
09564     std::chrono::time_point<std::chrono::steady_clock> start_time,
09565     Response &res, bool &success, Error &error) {
09566     success = true;
09567     Response proxy_res;
09568     if (!detail::process_client_socket(
09569         socket.sock, read_timeout_sec_, read_timeout_usec_,
09570         write_timeout_sec_, write_timeout_usec_, max_timeout_msec_,
09571         start_time, [&](Stream &strm) {
09572             Request req2;
09573             req2.method = "CONNECT";
09574             req2.path = host_and_port;
09575             if (max_timeout_msec_ > 0) {
09576                 req2.start_time_ = std::chrono::steady_clock::now();
09577             }
09578             return process_request(strm, req2, proxy_res, false, error);
09579         })) {
09580         // Thread-safe to close everything because we are assuming there are no
09581         // requests in flight
09582         shutdown_ssl(socket, true);
09583         shutdown_socket(socket);
09584         close_socket(socket);
09585         success = false;
09586         return false;
09587     }
09588
09589     if (proxy_res.status == StatusCode::ProxyAuthenticationRequired_407) {
09590         if (!proxy_digest_auth_username_.empty() &&
09591             !proxy_digest_auth_password_.empty()) {
09592             std::map<std::string, std::string> auth;
09593             if (detail::parse_www_authenticate(proxy_res, auth, true)) {
09594                 proxy_res = Response();
09595                 if (!detail::process_client_socket(
09596                     socket.sock, read_timeout_sec_, read_timeout_usec_,
09597                     write_timeout_sec_, write_timeout_usec_, max_timeout_msec_,
09598                     start_time, [&](Stream &strm) {
09599                         Request req3;
09600                         req3.method = "CONNECT";
09601                         req3.path = host_and_port;
09602                         req3.headers.insert(detail::make_digest_authentication_header(
09603                             req3, auth, 1, detail::random_string(10),
09604                             proxy_digest_auth_username_, proxy_digest_auth_password_,
09605                             true)));

```

```

09606         if (max_timeout_msec_ > 0) {
09607             req3.start_time_ = std::chrono::steady_clock::now();
09608         }
09609         return process_request(strm, req3, proxy_res, false, error);
09610     }));
09611     // Thread-safe to close everything because we are assuming there are
09612     // no requests in flight
09613     shutdown_ssl(socket, true);
09614     shutdown_socket(socket);
09615     close_socket(socket);
09616     success = false;
09617     return false;
09618 }
09619 }
09620 }
09621 }

09622 // If status code is not 200, proxy request is failed.
09623 // Set error to ProxyConnection and return proxy response
09624 // as the response of the request
09625 if (proxy_res.status != StatusCode::OK_200) {
09626     error = Error::ProxyConnection;
09627     res = std::move(proxy_res);
09628     // Thread-safe to close everything because we are assuming there are
09629     // no requests in flight
09630     shutdown_ssl(socket, true);
09631     shutdown_socket(socket);
09632     close_socket(socket);
09633     return false;
09634 }
09635 }

09636 return true;
09637 }

09638 }

09639 inline bool SSLClient::load_certs() {
09640     auto ret = true;
09641
09642     std::call_once(initialize_cert_, [&]{
09643         std::lock_guard<std::mutex> guard(ctx_mutex_);
09644         if (!ca_cert_file_path_.empty()) {
09645             if (!SSL_CTX_load_verify_locations(ctx_, ca_cert_file_path_.c_str(),
09646                                              nullptr)) {
09647                 ret = false;
09648             }
09649         }
09650     } else if (!ca_cert_dir_path_.empty()) {
09651         if (!SSL_CTX_load_verify_locations(ctx_, nullptr,
09652                                              ca_cert_dir_path_.c_str())) {
09653             ret = false;
09654         }
09655     } else {
09656         auto loaded = false;
09657 #ifdef _WIN32
09658         loaded =
09659             detail::load_system_certs_on_windows(SSL_CTX_get_cert_store(ctx_));
09660 #elif defined(CPPHTTPLIB_USE_CERTS_FROM_MACOSX_KEYCHAIN) && defined(__APPLE__)
09661 #if TARGET_OS_OSX
09662         loaded = detail::load_system_certs_on_macos(SSL_CTX_get_cert_store(ctx_));
09663 #endif // TARGET_OS_OSX
09664 #endif // __WIN32
09665         if (!loaded) { SSL_CTX_set_default_verify_paths(ctx_); }
09666     }
09667 });
09668
09669     return ret;
09670 }

09671
09672 inline bool SSLClient::initialize_ssl(Socket &socket, Error &error) {
09673     auto ssl = detail::ssl_new(
09674         socket.sock, ctx_, ctx_mutex_,
09675         [&](SSL *ssl2) {
09676             if (server_certificate_verification_) {
09677                 if (!load_certs()) {
09678                     error = Error::SSLLoadingCerts;
09679                     return false;
09680                 }
09681                 SSL_set_verify(ssl2, SSL_VERIFY_NONE, nullptr);
09682             }
09683
09684             if (!detail::ssl_connect_or_accept_nonblocking(
09685                 socket.sock, ssl2, SSL_connect, connection_timeout_sec_,
09686                 connection_timeout_usecs_)) {
09687                 error = Error::SSLConnection;
09688                 return false;
09689             }
09690
09691             if (server_certificate_verification_) {
09692                 auto verification_status = SSLVerifierResponse::NoDecisionMade;

```

```
09693
09694     if (server_certificate_verifier_) {
09695         verification_status = server_certificate_verifier_(ssl2);
09696     }
09697
09698     if (verification_status == SSLVerifierResponse::CertificateRejected) {
09699         error = Error::SSLServerVerification;
09700         return false;
09701     }
09702
09703     if (verification_status == SSLVerifierResponse::NoDecisionMade) {
09704         verify_result_ = SSL_get_verify_result(ssl2);
09705
09706         if (verify_result_ != X509_V_OK) {
09707             error = Error::SSLServerVerification;
09708             return false;
09709         }
09710
09711         auto server_cert = SSL_get1_peer_certificate(ssl2);
09712         auto se = detail::scope_exit([&] { X509_free(server_cert); });
09713
09714         if (server_cert == nullptr) {
09715             error = Error::SSLServerVerification;
09716             return false;
09717         }
09718
09719         if (server_hostname_verification_) {
09720             if (!verify_host(server_cert)) {
09721                 error = Error::SSLServerHostnameVerification;
09722                 return false;
09723             }
09724         }
09725     }
09726 }
09727
09728     return true;
09729 },
09730 [&](SSL *ssl2) {
09731 #if defined(OPENSSL_IS_BORINGSSL)
09732     SSL_set_tlsext_host_name(ssl2, host_.c_str());
09733 #else
09734     // NOTE: Direct call instead of using the OpenSSL macro to suppress
09735     // -Wold-style-cast warning
09736     SSL_ctrl(ssl2, SSL_CTRL_SET_TLSEXT_HOSTNAME, TLSEXT_NAMETYPE_host_name,
09737             static_cast<void*>(const_cast<char*>(host_.c_str())));
09738 #endif
09739     return true;
09740 });
09741
09742 if (ssl) {
09743     socket.ssl = ssl;
09744     return true;
09745 }
09746
09747 shutdown_socket(socket);
09748 close_socket(socket);
09749 return false;
09750 }
09751
09752 inline void SSLClient::shutdown_ssl(Socket &socket, bool shutdown_gracefully) {
09753     shutdown_ssl_impl(socket, shutdown_gracefully);
09754 }
09755
09756 inline void SSLClient::shutdown_ssl_impl(Socket &socket,
09757                                             bool shutdown_gracefully) {
09758     if (socket.sock == INVALID_SOCKET) {
09759         assert(socket.ssl == nullptr);
09760         return;
09761     }
09762     if (socket.ssl) {
09763         detail::ssl_delete(ctx_mutex_, socket.ssl, socket.sock,
09764                             shutdown_gracefully);
09765         socket.ssl = nullptr;
09766     }
09767     assert(socket.ssl == nullptr);
09768 }
09769
09770 inline bool SSLClient::process_socket(
09771     const Socket &socket,
09772     std::chrono::time_point<std::chrono::steady_clock> start_time,
09773     std::function<bool(Stream &strm)> callback) {
09774     assert(socket.ssl);
09775     return detail::process_client_socket_ssl(
09776         socket.ssl, socket.sock, read_timeout_sec_, read_timeout_usec_,
09777         write_timeout_sec_, write_timeout_usec_, max_timeout_msec_, start_time,
09778         std::move(callback));
09779 }
```

```
09780
09781 inline bool SSLClient::is_ssl() const { return true; }
09782
09783 inline bool SSLClient::verify_host(X509 *server_cert) const {
09784 /* Quote from RFC2818 section 3.1 "Server Identity"
09785
09786 If a subjectAltName extension of type dNSName is present, that MUST
09787 be used as the identity. Otherwise, the (most specific) Common Name
09788 field in the Subject field of the certificate MUST be used. Although
09789 the use of the Common Name is existing practice, it is deprecated and
09790 Certification Authorities are encouraged to use the dDNSName instead.
09791
09792 Matching is performed using the matching rules specified by
09793 [RFC2459]. If more than one identity of a given type is present in
09794 the certificate (e.g., more than one dNSName name, a match in any one
09795 of the set is considered acceptable.) Names may contain the wildcard
09796 character * which is considered to match any single domain name
09797 component or component fragment. E.g., *.a.com matches foo.a.com but
09798 not bar.foo.a.com. f*.com matches foo.com but not bar.com.
09799
09800 In some cases, the URI is specified as an IP address rather than a
09801 hostname. In this case, the iPAddress subjectAltName must be present
09802 in the certificate and must exactly match the IP in the URI.
09803
09804 */
09805 return verify_host_with_subject_alt_name(server_cert) ||
09806     verify_host_with_common_name(server_cert);
09807 }
09808
09809 inline bool
09810 SSLClient::verify_host_with_subject_alt_name(X509 *server_cert) const {
09811 auto ret = false;
09812
09813 auto type = GEN_DNS;
09814
09815 struct in6_addr addr6 = {};
09816 struct in_addr addr = {};
09817 size_t addr_len = 0;
09818
09819 #ifndef __MINGW32__
09820 if (inet_pton(AF_INET6, host_.c_str(), &addr6)) {
09821     type = GEN_IPADD;
09822     addr_len = sizeof(struct in6_addr);
09823 } else if (inet_pton(AF_INET, host_.c_str(), &addr)) {
09824     type = GEN_IPADD;
09825     addr_len = sizeof(struct in_addr);
09826 }
09827 #endif
09828
09829 auto alt_names = static_cast<const struct stack_st_GENERAL_NAME *>(
09830     X509_get_ext_d2i(server_cert, NID_subject_alt_name, nullptr, nullptr));
09831
09832 if (alt_names) {
09833     auto dsn_matched = false;
09834     auto ip_matched = false;
09835
09836     auto count = sk_GENERAL_NAME_num(alt_names);
09837
09838     for (decltype(count) i = 0; i < count && !dsn_matched; i++) {
09839         auto val = sk_GENERAL_NAME_value(alt_names, i);
09840         if (val->type == type) {
09841             auto name =
09842                 reinterpret_cast<const char *>(ASN1_STRING_get0_data(val->d.ia5));
09843             auto name_len = static_cast<size_t>(ASN1_STRING_length(val->d.ia5));
09844
09845             switch (type) {
09846                 case GEN_DNS: dsn_matched = check_host_name(name, name_len); break;
09847
09848                 case GEN_IPADD:
09849                     if (!memcmp(&addr6, name, addr_len) ||
09850                         !memcmp(&addr, name, addr_len)) {
09851                         ip_matched = true;
09852                     }
09853                     break;
09854             }
09855         }
09856     }
09857
09858     if (dsn_matched || ip_matched) { ret = true; }
09859 }
09860
09861 GENERAL_NAMES_free(const_cast<STACK_OF(GENERAL_NAME) *>(
09862     reinterpret_cast<const STACK_OF(GENERAL_NAME) *>(alt_names)));
09863
09864 return ret;
09865
09866 inline bool SSLClient::verify_host_with_common_name(X509 *server_cert) const {
```

```
09867 const auto subject_name = X509_get_subject_name(server_cert);
09868 if (subject_name != nullptr) {
09870     char name[BUFSIZ];
09871     auto name_len = X509_NAME_get_text_by_NID(subject_name, NID_commonName,
09872                                                 name, sizeof(name));
09873
09874     if (name_len != -1) {
09875         return check_host_name(name, static_cast<size_t>(name_len));
09876     }
09877 }
09878
09879 return false;
09880 }
09881
09882 inline bool SSLClient::check_host_name(const char *pattern,
09883                                         size_t pattern_len) const {
09884     if (host_.size() == pattern_len && host_ == pattern) { return true; }
09885
09886 // Wildcard match
09887 // https://bugs.launchpad.net/ubuntu/+source/firefox-3.0/+bug/376484
09888 std::vector<std::string> pattern_components;
09889 detail::split(&pattern[0], &pattern[pattern_len], ':',
09900     [&](const char *b, const char *e) {
09901         pattern_components.emplace_back(b, e);
09902     });
09903
09904 if (host_components_.size() != pattern_components.size()) { return false; }
09905
09906 auto itr = pattern_components.begin();
09907 for (const auto &h : host_components_) {
09908     auto &p = *itr;
09909     if (p != h && p != "*") {
09910         auto partial_match = (p.size() > 0 && p[p.size() - 1] == '*' &&
09911             !p.compare(0, p.size() - 1, h));
09912         if (!partial_match) { return false; }
09913     }
09914     ++itr;
09915 }
09916
09917 return true;
09918 }
09919 #endif
09920
09921 // Universal client implementation
09922 inline Client::Client(const std::string &scheme_host_port)
09923     : Client(scheme_host_port, std::string(), std::string()) {}
09924
09925 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
09926     if (!scheme.empty() && (scheme != "http" && scheme != "https")) {
09927 #else
09928     if (!scheme.empty() && scheme != "http") {
09929 #endif
09930 #ifndef CPPHTTPPLIB_NO_EXCEPTIONS
09931     std::string msg = "The " + scheme + " scheme is not supported.";
09932     throw std::invalid_argument(msg);
09933 #endif
09934     return;
09935 }
09936
09937 auto is_ssl = scheme == "https";
09938
09939 auto host = m[2].str();
09940 if (host.empty()) { host = m[3].str(); }
09941
09942 auto port_str = m[4].str();
09943 auto port = !port_str.empty() ? std::stoi(port_str) : (is_ssl ? 443 : 80);
09944
09945 if (is_ssl) {
09946 #ifdef CPPHTTPPLIB_OPENSSL_SUPPORT
09947     cli_ = detail::make_unique<SSLClient>(host, port, client_cert_path,
09948                                             client_key_path);
09949     is_ssl_ = is_ssl;
09950 #endif
09951 } else {
09952     cli_ = detail::make_unique<ClientImpl>(host, port, client_cert_path,
09953                                             client_key_path);
09954 }
```

```
09954     }
09955 } else {
09956 // NOTE: Update TEST(UniversalClientImplTest, Ipv6LiteralAddress)
09957 // if port param below changes.
09958 cli_ = detail::make_unique<ClientImpl>(scheme_host_port, 80,
09959                         client_cert_path, client_key_path);
09960 }
09961 } // namespace detail
09962
09963 inline Client::Client(const std::string &host, int port)
09964 : cli_(detail::make_unique<ClientImpl>(host, port)) {}
09965
09966 inline Client::Client(const std::string &host, int port,
09967             const std::string &client_cert_path,
09968             const std::string &client_key_path)
09969 : cli_(detail::make_unique<ClientImpl>(host, port, client_cert_path,
09970                         client_key_path)) {}
09971
09972 inline Client::~Client() = default;
09973
09974 inline bool Client::is_valid() const {
09975     return cli_ != nullptr && cli_->is_valid();
09976 }
09977
09978 inline Result Client::Get(const std::string &path) { return cli_->Get(path); }
09979 inline Result Client::Get(const std::string &path, const Headers &headers) {
09980     return cli_->Get(path, headers);
09981 }
09982 inline Result Client::Get(const std::string &path, Progress progress) {
09983     return cli_->Get(path, std::move(progress));
09984 }
09985 inline Result Client::Get(const std::string &path, const Headers &headers,
09986             Progress progress) {
09987     return cli_->Get(path, headers, std::move(progress));
09988 }
09989 inline Result Client::Get(const std::string &path,
09990             ContentReceiver content_receiver) {
09991     return cli_->Get(path, std::move(content_receiver));
09992 }
09993 inline Result Client::Get(const std::string &path, const Headers &headers,
09994             ContentReceiver content_receiver) {
09995     return cli_->Get(path, headers, std::move(content_receiver));
09996 }
09997 inline Result Client::Get(const std::string &path,
09998             ContentReceiver content_receiver, Progress progress) {
09999     return cli_->Get(path, std::move(content_receiver), std::move(progress));
10000 }
10001 inline Result Client::Get(const std::string &path, const Headers &headers,
10002             ContentReceiver content_receiver, Progress progress) {
10003     return cli_->Get(path, headers, std::move(content_receiver),
10004                         std::move(progress));
10005 }
10006 inline Result Client::Get(const std::string &path,
10007             ResponseHandler response_handler,
10008             ContentReceiver content_receiver) {
10009     return cli_->Get(path, std::move(response_handler),
10010                         std::move(content_receiver));
10011 }
10012 inline Result Client::Get(const std::string &path, const Headers &headers,
10013             ResponseHandler response_handler,
10014             ContentReceiver content_receiver) {
10015     return cli_->Get(path, headers, std::move(response_handler),
10016                         std::move(content_receiver));
10017 }
10018 inline Result Client::Get(const std::string &path,
10019             ResponseHandler response_handler,
10020             ContentReceiver content_receiver, Progress progress) {
10021     return cli_->Get(path, std::move(response_handler),
10022                         std::move(content_receiver), std::move(progress));
10023 }
10024 inline Result Client::Get(const std::string &path, const Headers &headers,
10025             ResponseHandler response_handler,
10026             ContentReceiver content_receiver, Progress progress) {
10027     return cli_->Get(path, headers, std::move(response_handler),
10028                         std::move(content_receiver), std::move(progress));
10029 }
10030 inline Result Client::Get(const std::string &path, const Params &params,
10031             const Headers &headers, Progress progress) {
10032     return cli_->Get(path, params, headers, std::move(progress));
10033 }
10034 inline Result Client::Get(const std::string &path, const Params &params,
10035             const Headers &headers,
10036             ContentReceiver content_receiver, Progress progress) {
10037     return cli_->Get(path, params, headers, std::move(content_receiver),
10038                         std::move(progress));
10039 }
10040 inline Result Client::Get(const std::string &path, const Params &params,
```

```
10041         const Headers &headers,
10042         ResponseHandler response_handler,
10043         ContentReceiver content_receiver, Progress progress) {
10044     return cli_->Get(path, params, headers, std::move(response_handler),
10045                     std::move(content_receiver), std::move(progress));
10046 }
10047
10048 inline Result Client::Head(const std::string &path) { return cli_->Head(path); }
10049 inline Result Client::Head(const std::string &path, const Headers &headers) {
10050     return cli_->Head(path, headers);
10051 }
10052
10053 inline Result Client::Post(const std::string &path) { return cli_->Post(path); }
10054 inline Result Client::Post(const std::string &path, const Headers &headers) {
10055     return cli_->Post(path, headers);
10056 }
10057 inline Result Client::Post(const std::string &path, const char *body,
10058                             size_t content_length,
10059                             const std::string &content_type) {
10060     return cli_->Post(path, body, content_length, content_type);
10061 }
10062 inline Result Client::Post(const std::string &path, const Headers &headers,
10063                             const char *body, size_t content_length,
10064                             const std::string &content_type) {
10065     return cli_->Post(path, headers, body, content_length, content_type);
10066 }
10067 inline Result Client::Post(const std::string &path, const Headers &headers,
10068                             const char *body, size_t content_length,
10069                             const std::string &content_type, Progress progress) {
10070     return cli_->Post(path, headers, body, content_length, content_type,
10071                         progress);
10072 }
10073 inline Result Client::Post(const std::string &path, const std::string &body,
10074                             const std::string &content_type) {
10075     return cli_->Post(path, body, content_type);
10076 }
10077 inline Result Client::Post(const std::string &path, const std::string &body,
10078                             const std::string &content_type, Progress progress) {
10079     return cli_->Post(path, body, content_type, progress);
10080 }
10081 inline Result Client::Post(const std::string &path, const Headers &headers,
10082                             const std::string &body,
10083                             const std::string &content_type) {
10084     return cli_->Post(path, headers, body, content_type);
10085 }
10086 inline Result Client::Post(const std::string &path, const Headers &headers,
10087                             const std::string &body,
10088                             const std::string &content_type, Progress progress) {
10089     return cli_->Post(path, headers, body, content_type, progress);
10090 }
10091 inline Result Client::Post(const std::string &path, size_t content_length,
10092                             ContentProvider content_provider,
10093                             const std::string &content_type) {
10094     return cli_->Post(path, content_length, std::move(content_provider),
10095                         content_type);
10096 }
10097 inline Result Client::Post(const std::string &path,
10098                             ContentProviderWithoutLength content_provider,
10099                             const std::string &content_type) {
10100    return cli_->Post(path, std::move(content_provider), content_type);
10101 }
10102 inline Result Client::Post(const std::string &path, const Headers &headers,
10103                             size_t content_length,
10104                             ContentProvider content_provider,
10105                             const std::string &content_type) {
10106    return cli_->Post(path, headers, content_length, std::move(content_provider),
10107                         content_type);
10108 }
10109 inline Result Client::Post(const std::string &path, const Headers &headers,
10110                             ContentProviderWithoutLength content_provider,
10111                             const std::string &content_type) {
10112    return cli_->Post(path, headers, std::move(content_provider), content_type);
10113 }
10114 inline Result Client::Post(const std::string &path, const Params &params) {
10115    return cli_->Post(path, params);
10116 }
10117 inline Result Client::Post(const std::string &path, const Headers &headers,
10118                             const Params &params) {
10119    return cli_->Post(path, headers, params);
10120 }
10121 inline Result Client::Post(const std::string &path, const Headers &headers,
10122                             const Params &params, Progress progress) {
10123    return cli_->Post(path, headers, params, progress);
10124 }
10125 inline Result Client::Post(const std::string &path,
10126                             const MultipartFormDataItems &items) {
10127    return cli_->Post(path, items);
```

```
10128 }
10129 inline Result Client::Post(const std::string &path, const Headers &headers,
10130                         const MultipartFormDataItems &items) {
10131     return cli_->Post(path, headers, items);
10132 }
10133 inline Result Client::Post(const std::string &path, const Headers &headers,
10134                         const MultipartFormDataItems &items,
10135                         const std::string &boundary) {
10136     return cli_->Post(path, headers, items, boundary);
10137 }
10138 inline Result
10139 Client::Post(const std::string &path, const Headers &headers,
10140                 const MultipartFormDataItems &items,
10141                 const MultipartFormDataProviderItems &provider_items) {
10142     return cli_->Post(path, headers, items, provider_items);
10143 }
10144 inline Result Client::Put(const std::string &path) { return cli_->Put(path); }
10145 inline Result Client::Put(const std::string &path, const char *body,
10146                           size_t content_length,
10147                           const std::string &content_type) {
10148     return cli_->Put(path, body, content_length, content_type);
10149 }
10150 inline Result Client::Put(const std::string &path, const Headers &headers,
10151                           const char *body, size_t content_length,
10152                           const std::string &content_type) {
10153     return cli_->Put(path, headers, body, content_length, content_type);
10154 }
10155 inline Result Client::Put(const std::string &path, const Headers &headers,
10156                           const char *body, size_t content_length,
10157                           const std::string &content_type, Progress progress) {
10158     return cli_->Put(path, headers, body, content_length, content_type, progress);
10159 }
10160 inline Result Client::Put(const std::string &path, const std::string &body,
10161                           const std::string &content_type) {
10162     return cli_->Put(path, body, content_type);
10163 }
10164 inline Result Client::Put(const std::string &path, const std::string &body,
10165                           const std::string &content_type, Progress progress) {
10166     return cli_->Put(path, body, content_type, progress);
10167 }
10168 inline Result Client::Put(const std::string &path, const Headers &headers,
10169                           const std::string &body,
10170                           const std::string &content_type) {
10171     return cli_->Put(path, headers, body, content_type);
10172 }
10173 inline Result Client::Put(const std::string &path, const Headers &headers,
10174                           const std::string &body,
10175                           const std::string &content_type, Progress progress) {
10176     return cli_->Put(path, headers, body, content_type, progress);
10177 }
10178 inline Result Client::Put(const std::string &path, size_t content_length,
10179                           ContentProvider content_provider,
10180                           const std::string &content_type) {
10181     return cli_->Put(path, content_length, std::move(content_provider),
10182                       content_type);
10183 }
10184 inline Result Client::Put(const std::string &path,
10185                           ContentProviderWithoutLength content_provider,
10186                           const std::string &content_type) {
10187     return cli_->Put(path, std::move(content_provider), content_type);
10188 }
10189 inline Result Client::Put(const std::string &path, const Headers &headers,
10190                           size_t content_length,
10191                           ContentProvider content_provider,
10192                           const std::string &content_type) {
10193     return cli_->Put(path, headers, content_length, std::move(content_provider),
10194                       content_type);
10195 }
10196 inline Result Client::Put(const std::string &path, const Headers &headers,
10197                           ContentProviderWithoutLength content_provider,
10198                           const std::string &content_type) {
10199     return cli_->Put(path, headers, std::move(content_provider), content_type);
10200 }
10201 inline Result Client::Put(const std::string &path, const Params &params) {
10202     return cli_->Put(path, params);
10203 }
10204 inline Result Client::Put(const std::string &path, const Headers &headers,
10205                           const Params &params) {
10206     return cli_->Put(path, headers, params);
10207 }
10208 inline Result Client::Put(const std::string &path, const Headers &headers,
10209                           const Params &params, Progress progress) {
10210     return cli_->Put(path, headers, params, progress);
10211 }
10212 inline Result Client::Put(const std::string &path,
10213                           const MultipartFormDataItems &items) {
10214     return cli_->Put(path, items);
```

```
10215 }
10216 inline Result Client::Put(const std::string &path, const Headers &headers,
10217         const MultipartFormDataItems &items) {
10218     return cli_->Put(path, headers, items);
10219 }
10220 inline Result Client::Put(const std::string &path, const Headers &headers,
10221         const MultipartFormDataItems &items,
10222         const std::string &boundary) {
10223     return cli_->Put(path, headers, items, boundary);
10224 }
10225 inline Result
10226 Client::Put(const std::string &path, const Headers &headers,
10227         const MultipartFormDataItems &items,
10228         const MultipartFormDataProviderItems &provider_items) {
10229     return cli_->Put(path, headers, items, provider_items);
10230 }
10231 inline Result Client::Patch(const std::string &path) {
10232     return cli_->Patch(path);
10233 }
10234 inline Result Client::Patch(const std::string &path, const char *body,
10235         size_t content_length,
10236         const std::string &content_type) {
10237     return cli_->Patch(path, body, content_length, content_type);
10238 }
10239 inline Result Client::Patch(const std::string &path, const char *body,
10240         size_t content_length,
10241         const std::string &content_type,
10242         Progress progress) {
10243     return cli_->Patch(path, body, content_length, content_type, progress);
10244 }
10245 inline Result Client::Patch(const std::string &path, const Headers &headers,
10246         const char *body, size_t content_length,
10247         const std::string &content_type) {
10248     return cli_->Patch(path, headers, body, content_length, content_type);
10249 }
10250 inline Result Client::Patch(const std::string &path, const Headers &headers,
10251         const char *body, size_t content_length,
10252         const std::string &content_type,
10253         Progress progress) {
10254     return cli_->Patch(path, headers, body, content_length, content_type,
10255         progress);
10256 }
10257 inline Result Client::Patch(const std::string &path, const std::string &body,
10258         const std::string &content_type) {
10259     return cli_->Patch(path, body, content_type);
10260 }
10261 inline Result Client::Patch(const std::string &path, const std::string &body,
10262         const std::string &content_type,
10263         Progress progress) {
10264     return cli_->Patch(path, body, content_type, progress);
10265 }
10266 inline Result Client::Patch(const std::string &path, const Headers &headers,
10267         const std::string &body,
10268         const std::string &content_type) {
10269     return cli_->Patch(path, headers, body, content_type);
10270 }
10271 inline Result Client::Patch(const std::string &path, const Headers &headers,
10272         const std::string &body,
10273         const std::string &content_type,
10274         Progress progress) {
10275     return cli_->Patch(path, headers, body, content_type, progress);
10276 }
10277 inline Result Client::Patch(const std::string &path, size_t content_length,
10278         ContentProvider content_provider,
10279         const std::string &content_type) {
10280     return cli_->Patch(path, content_length, std::move(content_provider),
10281         content_type);
10282 }
10283 inline Result Client::Patch(const std::string &path,
10284         ContentProviderWithoutLength content_provider,
10285         const std::string &content_type) {
10286     return cli_->Patch(path, std::move(content_provider), content_type);
10287 }
10288 inline Result Client::Patch(const std::string &path, const Headers &headers,
10289         size_t content_length,
10290         ContentProvider content_provider,
10291         const std::string &content_type) {
10292     return cli_->Patch(path, headers, content_length, std::move(content_provider),
10293         content_type);
10294 }
10295 inline Result Client::Patch(const std::string &path, const Headers &headers,
10296         ContentProviderWithoutLength content_provider,
10297         const std::string &content_type) {
10298     return cli_->Patch(path, headers, std::move(content_provider), content_type);
10299 }
10300 inline Result Client::Delete(const std::string &path) {
10301     return cli_->Delete(path);
```

```
10302 }
10303 inline Result Client::Delete(const std::string &path, const Headers &headers) {
10304     return cli_->Delete(path, headers);
10305 }
10306 inline Result Client::Delete(const std::string &path, const char *body,
10307                                 size_t content_length,
10308                                 const std::string &content_type) {
10309     return cli_->Delete(path, body, content_length, content_type);
10310 }
10311 inline Result Client::Delete(const std::string &path, const char *body,
10312                                 size_t content_length,
10313                                 const std::string &content_type,
10314                                 Progress progress) {
10315     return cli_->Delete(path, body, content_length, content_type, progress);
10316 }
10317 inline Result Client::Delete(const std::string &path, const Headers &headers,
10318                                 const char *body, size_t content_length,
10319                                 const std::string &content_type) {
10320     return cli_->Delete(path, headers, body, content_length, content_type);
10321 }
10322 inline Result Client::Delete(const std::string &path, const Headers &headers,
10323                                 const char *body, size_t content_length,
10324                                 const std::string &content_type,
10325                                 Progress progress) {
10326     return cli_->Delete(path, headers, body, content_length, content_type,
10327                           progress);
10328 }
10329 inline Result Client::Delete(const std::string &path, const std::string &body,
10330                                 const std::string &content_type) {
10331     return cli_->Delete(path, body, content_type);
10332 }
10333 inline Result Client::Delete(const std::string &path, const std::string &body,
10334                                 const std::string &content_type,
10335                                 Progress progress) {
10336     return cli_->Delete(path, body, content_type, progress);
10337 }
10338 inline Result Client::Delete(const std::string &path, const Headers &headers,
10339                                 const std::string &body,
10340                                 const std::string &content_type) {
10341     return cli_->Delete(path, headers, body, content_type);
10342 }
10343 inline Result Client::Delete(const std::string &path, const Headers &headers,
10344                                 const std::string &body,
10345                                 const std::string &content_type,
10346                                 Progress progress) {
10347     return cli_->Delete(path, headers, body, content_type, progress);
10348 }
10349 inline Result Client::Options(const std::string &path) {
10350     return cli_->Options(path);
10351 }
10352 inline Result Client::Options(const std::string &path, const Headers &headers) {
10353     return cli_->Options(path, headers);
10354 }
10355
10356 inline bool Client::send(Request &req, Response &res, Error &error) {
10357     return cli_->send(req, res, error);
10358 }
10359
10360 inline Result Client::send(const Request &req) { return cli_->send(req); }
10361
10362 inline void Client::stop() { cli_->stop(); }
10363
10364 inline std::string Client::host() const { return cli_->host(); }
10365
10366 inline int Client::port() const { return cli_->port(); }
10367
10368 inline size_t Client::is_socket_open() const { return cli_->is_socket_open(); }
10369
10370 inline socket_t Client::socket() const { return cli_->socket(); }
10371
10372 inline void
10373 Client::set_hostname_map(std::map<std::string, std::string> addr_map) {
10374     cli_->set_hostname_map(std::move(addr_map));
10375 }
10376
10377 inline void Client::set_default_headers(Headers headers) {
10378     cli_->set_default_headers(std::move(headers));
10379 }
10380
10381 inline void Client::set_header_writer(
10382     std::function<ssize_t(Stream &, Headers &)> const &writer) {
10383     cli_->set_header_writer(writer);
10384 }
10385
10386 inline void Client::set_address_family(int family) {
10387     cli_->set_address_family(family);
10388 }
```

```
10389
10390 inline void Client::set_tcp_nodelay(bool on) { cli_->set_tcp_nodelay(on); }
10391
10392 inline void Client::set_socket_options(SocketOptions socket_options) {
10393   cli_->set_socket_options(std::move(socket_options));
10394 }
10395
10396 inline void Client::set_connection_timeout(time_t sec, time_t usec) {
10397   cli_->set_connection_timeout(sec, usec);
10398 }
10399
10400 inline void Client::set_read_timeout(time_t sec, time_t usec) {
10401   cli_->set_read_timeout(sec, usec);
10402 }
10403
10404 inline void Client::set_write_timeout(time_t sec, time_t usec) {
10405   cli_->set_write_timeout(sec, usec);
10406 }
10407
10408 inline void Client::set_basic_auth(const std::string &username,
10409                           const std::string &password) {
10410   cli_->set_basic_auth(username, password);
10411 }
10412 inline void Client::set_bearer_token_auth(const std::string &token) {
10413   cli_->set_bearer_token_auth(token);
10414 }
10415 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
10416 inline void Client::set_digest_auth(const std::string &username,
10417                           const std::string &password) {
10418   cli_->set_digest_auth(username, password);
10419 }
10420 #endif
10421
10422 inline void Client::set_keep_alive(bool on) { cli_->set_keep_alive(on); }
10423 inline void Client::set_follow_location(bool on) {
10424   cli_->set_follow_location(on);
10425 }
10426
10427 inline void Client::set_url_encode(bool on) { cli_->set_url_encode(on); }
10428
10429 inline void Client::set_compress(bool on) { cli_->set_compress(on); }
10430
10431 inline void Client::set_decompress(bool on) { cli_->set_decompress(on); }
10432
10433 inline void Client::set_interface(const std::string &intf) {
10434   cli_->set_interface(intf);
10435 }
10436
10437 inline void Client::set_proxy(const std::string &host, int port) {
10438   cli_->set_proxy(host, port);
10439 }
10440 inline void Client::set_proxy_basic_auth(const std::string &username,
10441                           const std::string &password) {
10442   cli_->set_proxy_basic_auth(username, password);
10443 }
10444 inline void Client::set_proxy_bearer_token_auth(const std::string &token) {
10445   cli_->set_proxy_bearer_token_auth(token);
10446 }
10447 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
10448 inline void Client::set_proxy_digest_auth(const std::string &username,
10449                           const std::string &password) {
10450   cli_->set_proxy_digest_auth(username, password);
10451 }
10452 #endif
10453
10454 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
10455 inline void Client::enable_server_certificate_verification(bool enabled) {
10456   cli_->enable_server_certificate_verification(enabled);
10457 }
10458
10459 inline void Client::enable_server_hostname_verification(bool enabled) {
10460   cli_->enable_server_hostname_verification(enabled);
10461 }
10462
10463 inline void Client::set_server_certificate_verifier(
10464   std::function<SSLVerifierResponse(SSL *ssl)> verifier) {
10465   cli_->set_server_certificate_verifier(verifier);
10466 }
10467 #endif
10468
10469 inline void Client::set_logger(Logger logger) {
10470   cli_->set_logger(std::move(logger));
10471 }
10472
10473 #ifdef CPPHTTPLIB_OPENSSL_SUPPORT
10474 inline void Client::set_ca_cert_path(const std::string &ca_cert_file_path,
10475                           const std::string &ca_cert_dir_path) {
```

```

10476   cli_->set_ca_cert_path(ca_cert_file_path, ca_cert_dir_path);
10477 }
10478
10479 inline void Client::set_ca_cert_store(X509_STORE *ca_cert_store) {
10480   if (is_ssl_) {
10481     static_cast<SSLClient &>(*cli_).set_ca_cert_store(ca_cert_store);
10482   } else {
10483     cli_->set_ca_cert_store(ca_cert_store);
10484   }
10485 }
10486
10487 inline void Client::load_ca_cert_store(const char *ca_cert, std::size_t size) {
10488   set_ca_cert_store(cli_->create_ca_cert_store(ca_cert, size));
10489 }
10490
10491 inline long Client::get_openssl_verify_result() const {
10492   if (is_ssl_) {
10493     return static_cast<SSLClient &>(*cli_).get_openssl_verify_result();
10494   }
10495   return -1; // NOTE: -1 doesn't match any of X509_V_ERR_???
10496 }
10497
10498 inline SSL_CTX *Client::ssl_context() const {
10499   if (is_ssl_) { return static_cast<SSLClient &>(*cli_).ssl_context(); }
10500   return nullptr;
10501 }
10502 #endif
10503
10504 // -----
10505
10506 } // namespace httplib
10507
10508 #endif // CPPHTTPLIB_H

```

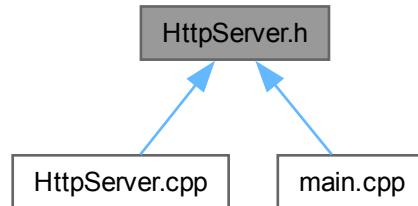
7.9 Файл HttpServer.h

#include "extern/httplib.h"

Граф включаемых заголовочных файлов для HttpServer.h:



Граф файлов, в которые включается этот файл:



Классы

- class [HttpServer](#)

Класс для запуска HTTP-сервера авторизации.

7.10 HttpServer.h

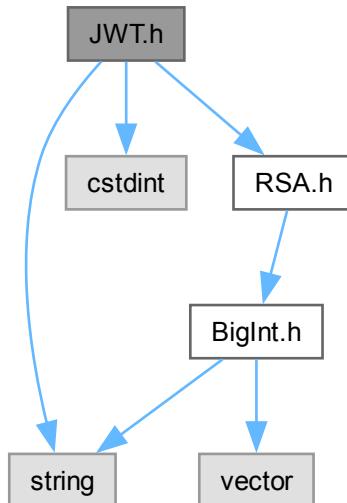
[См. документацию.](#)

```
00001 #pragma once
00002 #include "extern/httplib.h"
00003
0016 class HttpServer {
0017 public:
0026     static void start(int port = 8080);
0027 };
```

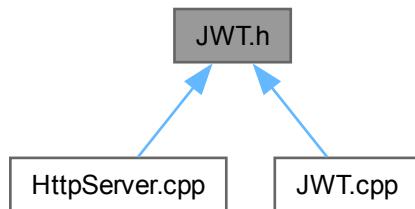
7.11 Файл JWT.h

```
#include <string>
#include <cstdint>
#include "RSA.h"
```

Граф включаемых заголовочных файлов для JWT.h:



Граф файлов, в которые включается этот файл:



Классы

- class [JWT](#)

Класс, реализующий создание и проверку JSON Web Token ([JWT](#)).

7.12 JWT.h

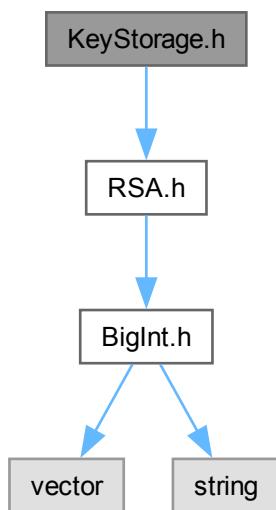
[См. документацию.](#)

```
00001 #pragma once
00002 #include <string>
00003 #include <cstdint>
00004 #include "RSA.h"
00005
00021 class JWT {
00022 public:
00040     static std::string createAccessToken(const std::string& subject,
00041                                         uint64_t expirationSeconds,
00042                                         const RSAPrivateKey& privKey);
00043
00060     static bool verifyAccessToken(const std::string& token,
00061                                     const RSAPublicKey& pubKey,
00062                                     std::string& outSubject);
00063
00076     static std::string createRefreshToken(const std::string& subject,
00077                                         uint64_t expirationSeconds,
00078                                         const RSAPrivateKey& privKey);
00079
00090     static bool verifyRefreshToken(const std::string& token,
00091                                     const RSAPublicKey& pubKey,
00092                                     std::string& outSubject);
00093 };
```

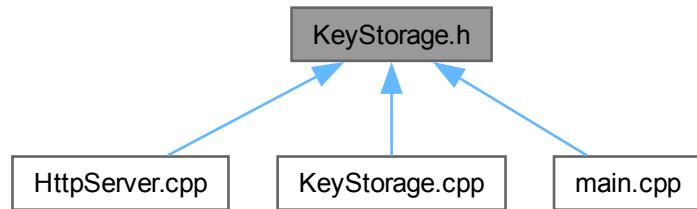
7.13 Файл KeyStorage.h

```
#include "RSA.h"
```

Граф включаемых заголовочных файлов для KeyStorage.h:



Граф файлов, в которые включается этот файл:



Классы

- class [KeyStorage](#)

Класс для хранения и загрузки RSA-ключей на диск.

7.14 KeyStorage.h

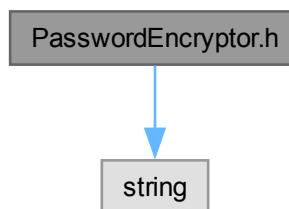
[См. документацию.](#)

```
00001 #pragma once
00002 #include "RSA.h"
00003
00025 class KeyStorage {
00026 public:
00035     static bool loadKeys(RSAPublicKey& publicKey, RSAPrivatekey& privKey);
00036
00046     static void saveKeys(const RSAPublicKey& publicKey, const RSAPrivatekey& privKey);
00047 };
```

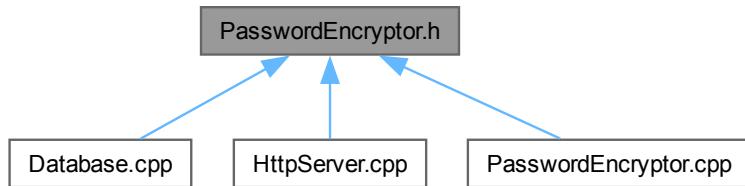
7.15 Файл PasswordEncryptor.h

#include <string>

Граф включаемых заголовочных файлов для PasswordEncryptor.h:



Граф файлов, в которые включается этот файл:



Классы

- class [PasswordEncryptor](#)
Утилита для безопасного хеширования паролей.

7.16 PasswordEncryptor.h

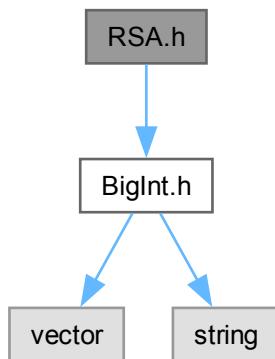
[См. документацию.](#)

```
00001 #pragma once
00002 #include <string>
00003
00017 class PasswordEncryptor {
00018 public:
00025     static std::string hashPassword(const std::string& password);
00026 };
```

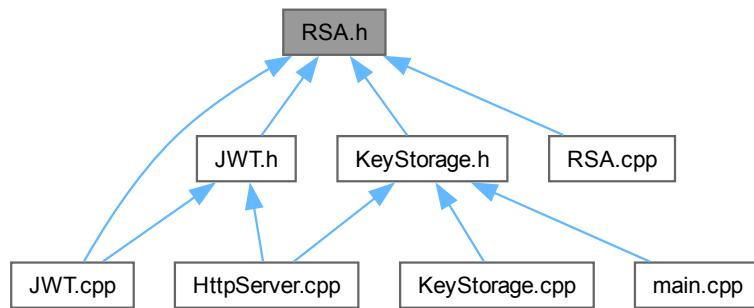
7.17 Файл RSA.h

```
#include "BigInt.h"
```

Граф включаемых заголовочных файлов для RSA.h:



Граф файлов, в которые включается этот файл:



Классы

- struct [RSA PublicKey](#)
Структура для хранения открытого (публичного) ключа [RSA](#).
- struct [RSA PrivateKey](#)
Структура для хранения закрытого (приватного) ключа [RSA](#).
- class [RSA](#)
Класс, реализующий основные операции алгоритма [RSA](#).

7.18 RSA.h

[См. документацию.](#)

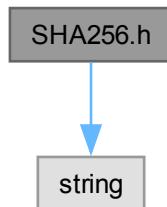
```

00001 #pragma once
00002 #include "BigInt.h"
00003
00011 struct RSA PublicKey {
00012     BigInt e;
00013     BigInt n;
00014 };
00015
00023 struct RSA PrivateKey {
00024     BigInt d;
00025     BigInt n;
00026 };
00027
00033 class RSA {
00034 public:
00049     static void generate_keys(RSA PublicKey& pub, RSA PrivateKey& priv, int bit_length = 64);
00050
00060     static BigInt encrypt(const BigInt& message, const RSA PublicKey& key);
00061
00071     static BigInt decrypt(const BigInt& cipher, const RSA PrivateKey& key);
00072
00082     static BigInt sign(const BigInt& hash, const RSA PrivateKey& key);
00083
00096     static bool verify(const std::string& messageHashHex,
00097                         const BigInt& signature,
00098                         const RSA PublicKey& key);
00099 };
  
```

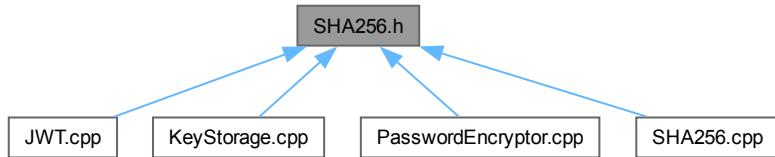
7.19 Файл SHA256.h

```
#include <string>
```

Граф включаемых заголовочных файлов для SHA256.h:



Граф файлов, в которые включается этот файл:



Классы

- class **SHA256**

Реализация криптографического хеш-функции SHA-256.

7.20 SHA256.h

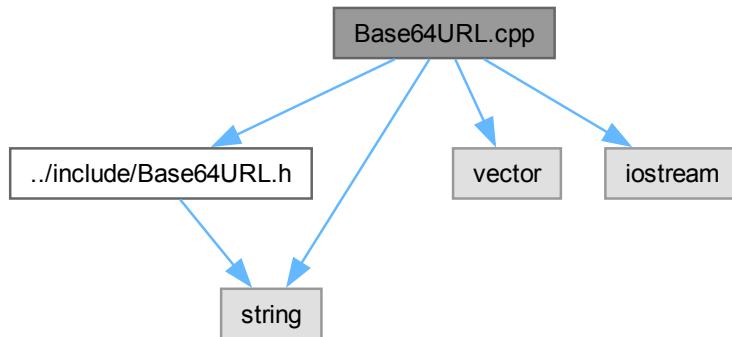
[См. документацию.](#)

```
00001 #pragma once
00002 #include <string>
00003
00017 class SHA256 {
00018 public:
00025     static std::string hash(const std::string& input);
00026 };
```

7.21 Файл Base64URL.cpp

```
#include "../include/Base64URL.h"
#include <string>
#include <vector>
#include <iostream>
```

Граф включаемых заголовочных файлов для Base64URL.cpp:



Переменные

- static const char * **base64_chars**

7.21.1 Переменные

7.21.1.1 base64_chars

```
const char* base64_chars [static]
```

Инициализатор

```
=
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklmnopqrstuvwxyz"
"0123456789+/"
```

См. определение в файле [Base64URL.cpp](#) строка 6

7.22 Base64URL.cpp

[См. документацию.](#)

```
00001 #include "../include/Base64URL.h"
00002 #include <string>
00003 #include <vector>
00004 #include <iostream>
00005
00006 static const char* base64_chars =
00007     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
00008     "abcdefghijklmnopqrstuvwxyz"
00009     "0123456789+/"
```

```

00008     "abcdefghijklmnopqrstuvwxyz"
00009     "0123456789+/";
00010
00011     std::string Base64URL::encode(const std::string& input) {
00012         std::cout << "[Base64URL::encode] Входная строка: " << input << std::endl;
00013
00014         std::string encoded;
00015         int val = 0, valb = -6;
00016         for (unsigned char c : input) {
00017             val = (val << 8) + c;
00018             valb += 8;
00019             while (valb >= 0) {
00020                 char ch = base64_chars[(val >> valb) & 0x3F];
00021                 encoded.push_back(ch);
00022                 valb -= 6;
00023             }
00024         }
00025
00026         if (valb > -6) {
00027             char ch = base64_chars[((val << 8) >> (valb + 8)) & 0x3F];
00028             encoded.push_back(ch);
00029         }
00030
00031         // Преобразование в Base64URL
00032         for (char& c : encoded) {
00033             if (c == '+') c = '_';
00034             else if (c == '/') c = '_';
00035         }
00036
00037         while (!encoded.empty() && encoded.back() == '=')
00038             encoded.pop_back();
00039
00040         std::cout << "[Base64URL::encode] Кодированная строка (Base64URL): " << encoded << std::endl;
00041         return encoded;
00042     }
00043
00044     std::string Base64URL::decode(const std::string& input) {
00045         std::cout << "[Base64URL::decode] Входная строка (Base64URL): " << input << std::endl;
00046
00047         std::string b64 = input;
00048         for (char& c : b64) {
00049             if (c == '_') c = '+';
00050             else if (c == '_') c = '/';
00051         }
00052
00053         while (b64.size() % 4 != 0)
00054             b64 += '=';
00055
00056         std::vector<int> T(256, -1);
00057         for (int i = 0; i < 64; i++) T[base64_chars[i]] = i;
00058
00059         std::string out;
00060         int val = 0, valb = -8;
00061         for (unsigned char c : b64) {
00062             if (T[c] == -1) break;
00063             val = (val << 6) + T[c];
00064             valb += 6;
00065             if (valb >= 0) {
00066                 char ch = char((val >> valb) & 0xFF);
00067                 out.push_back(ch);
00068                 valb -= 8;
00069             }
00070         }
00071
00072         std::cout << "[Base64URL::decode] Декодированная строка: " << out << std::endl;
00073         return out;
00074     }

```

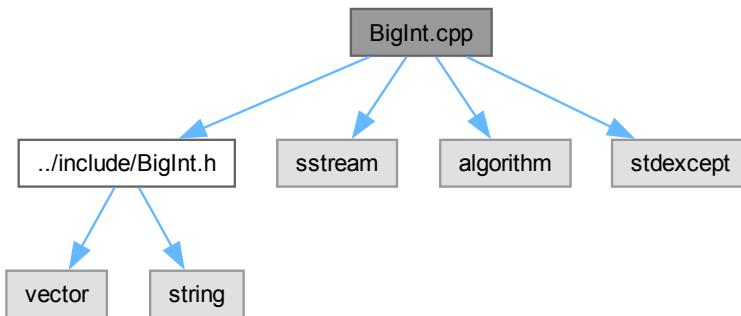
7.23 Файл BigInt.cpp

```

#include "../include/BigInt.h"
#include <iostream>
#include <algorithm>
#include <stdexcept>

```

Граф включаемых заголовочных файлов для BigInt.cpp:



7.24 BigInt.cpp

[См. документацию.](#)

```

00001 #include "../include/BigInt.h"
00002 #include <iostream>
00003 #include <algorithm>
00004 #include <stdexcept>
00005
00006 BigInt::BigInt() : digits{0}, negative(false) {}
00007
00008 BigInt::BigInt(int value) {
00009     if (value < 0) {
00010         negative = true;
00011         value = -value;
00012     }
00013     do {
00014         digits.push_back(value % 10);
00015         value /= 10;
00016     } while (value > 0);
00017 }
00018
00019 BigInt::BigInt(const std::string& str) {
00020     negative = !str.empty() && str[0] == '-';
00021     int start = negative ? 1 : 0;
00022     for (int i = static_cast<int>(str.length()) - 1; i >= start; --i) {
00023         if (isdigit(str[i]))
00024             digits.push_back(str[i] - '0');
00025     }
00026     if (digits.empty()) digits.push_back(0);
00027     trim();
00028 }
00029
00030 BigInt::BigInt(const std::string& str, int base) {
00031     if (base != 10 && base != 16) {
00032         throw std::invalid_argument("Unsupported base");
00033     }
00034
00035     std::string s = str;
00036     negative = false;
00037
00038     if (!s.empty() && s[0] == '+') {
00039         negative = true;
00040         s = s.substr(1);
00041     }
00042
00043     if (base == 10) {
00044         // обычный десятичный парсинг, переиспользуем текущий конструктор
00045         *this = BigInt(s);
00046         if (negative) *this = -(*this);
00047         return;
00048     }
00049     // парсинг hex
  
```

```

00051     BigInt result;
00052     BigInt basePow(1);
00053
00054     for (auto it = s.rbegin(); it != s.rend(); ++it) {
00055         char c = *it;
00056         int value = 0;
00057
00058         if (c >= '0' && c <= '9') value = c - '0';
00059         else if (c >= 'a' && c <= 'f') value = 10 + (c - 'a');
00060         else if (c >= 'A' && c <= 'F') value = 10 + (c - 'A');
00061         else throw std::invalid_argument("Invalid character in hex string");
00062
00063         result = result + basePow * BigInt(value);
00064         basePow = basePow * 16;
00065     }
00066
00067     if (negative) result = -result;
00068
00069     *this = result;
00070 }
00071
00072 std::string BigInt::toString() const {
00073     std::ostringstream oss;
00074     if (negative && isZero()) oss << "-";
00075     for (auto it = digits.rbegin(); it != digits.rend(); ++it)
00076         oss << *it;
00077     return oss.str();
00078 }
00079
00080 std::string BigInt::toString(int base) const {
00081     if (base != 10 && base != 16)
00082         throw std::invalid_argument("Unsupported base");
00083
00084     if (isZero()) return "0";
00085
00086     BigInt temp = *this;
00087     temp.negative = false;
00088
00089     std::string result;
00090     BigInt b(base);
00091
00092     while (!temp.isZero()) {
00093         BigInt digit = temp % b;
00094         int d = std::stoi(digit.toString());
00095
00096         char c;
00097         if (d < 10) c = '0' + d;
00098         else c = 'a' + (d - 10);
00099
00100         result += c;
00101         temp = temp / b;
00102     }
00103
00104     if (negative) result += '-';
00105     std::reverse(result.begin(), result.end());
00106     return result;
00107 }
00108
00109 void BigInt::trim() {
00110     while (digits.size() > 1 && digits.back() == 0)
00111         digits.pop_back();
00112     if (digits.size() == 1 && digits[0] == 0)
00113         negative = false;
00114 }
00115
00116 bool BigInt::isZero() const {
00117     return digits.size() == 1 && digits[0] == 0;
00118 }
00119
00120 bool BigInt::isNegative() const {
00121     return negative;
00122 }
00123
00124 int BigInt::compareAbs(const BigInt& a, const BigInt& b) {
00125     if (a.digits.size() != b.digits.size())
00126         return a.digits.size() < b.digits.size() ? -1 : 1;
00127     for (int i = static_cast<int>(a.digits.size()) - 1; i >= 0; --i) {
00128         if (a.digits[i] != b.digits[i])
00129             return a.digits[i] < b.digits[i] ? -1 : 1;
00130     }
00131     return 0;
00132 }
00133
00134 // Операторы сравнения
00135
00136 bool BigInt::operator==(const BigInt& other) const {
00137     return negative == other.negative && digits == other.digits;

```

```

00138 }
00139
00140 bool BigInt::operator!=(const BigInt& other) const {
00141     return !(*this == other);
00142 }
00143
00144 bool BigInt::operator<(const BigInt& other) const {
00145     if (negative != other.negative)
00146         return negative;
00147     int cmp = compareAbs(*this, other);
00148     return negative ? cmp > 0 : cmp < 0;
00149 }
00150
00151 bool BigInt::operator>(const BigInt& other) const {
00152     return other < *this;
00153 }
00154
00155 bool BigInt::operator<=(const BigInt& other) const {
00156     return !(*this > other);
00157 }
00158
00159 bool BigInt::operator>=(const BigInt& other) const {
00160     return !(*this < other);
00161 }
00162
00163 BigInt BigInt::operator-() const {
00164     BigInt result = *this;
00165     if (!isZero()) result.negative = !negative;
00166     return result;
00167 }
00168
00169 // Арифметика
00170
00171 BigInt BigInt::operator+(const BigInt& other) const {
00172     if (negative == other.negative) {
00173         BigInt result;
00174         result.negative = negative;
00175         result.digits.clear();
00176
00177         int carry = 0;
00178         size_t n = std::max(digits.size(), other.digits.size());
00179         for (size_t i = 0; i < n || carry; ++i) {
00180             int sum = carry;
00181             if (i < digits.size()) sum += digits[i];
00182             if (i < other.digits.size()) sum += other.digits[i];
00183             result.digits.push_back(sum % 10);
00184             carry = sum / 10;
00185         }
00186
00187         result.trim();
00188         return result;
00189     }
00190     return *this - (-other);
00191 }
00192
00193 BigInt BigInt::operator-(const BigInt& other) const {
00194     if (negative != other.negative) {
00195         return *this + (-other);
00196     }
00197
00198     if (compareAbs(*this, other) < 0) {
00199         BigInt result = other - *this;
00200         result.negative = !negative;
00201         return result;
00202     }
00203
00204     BigInt result;
00205     result.digits.clear();
00206     result.negative = negative;
00207
00208     int borrow = 0;
00209     for (size_t i = 0; i < digits.size(); ++i) {
00210         int diff = digits[i] - borrow;
00211         if (i < other.digits.size()) diff -= other.digits[i];
00212         if (diff < 0) {
00213             diff += 10;
00214             borrow = 1;
00215         } else {
00216             borrow = 0;
00217         }
00218         result.digits.push_back(diff);
00219     }
00220
00221     result.trim();
00222     return result;
00223 }
00224

```

```

00225 BigInt BigInt::operator*(const BigInt& other) const {
00226     BigInt result;
00227     result.digits.assign(digits.size() + other.digits.size(), 0);
00228     result.negative != other.negative;
00229
00230     for (size_t i = 0; i < digits.size(); ++i) {
00231         int carry = 0;
00232         for (size_t j = 0; j < other.digits.size() || carry; ++j) {
00233             int64_t cur = result.digits[i + j] +
00234                 digits[i] * 1LL * (j < other.digits.size() ? other.digits[j] : 0) + carry;
00235             result.digits[i + j] = cur % 10;
00236             carry = cur / 10;
00237         }
00238     }
00239
00240     result.trim();
00241     return result;
00242 }
00243
00244 BigInt BigInt::operator/(const BigInt& other) const {
00245     if (other.isZero()) throw std::domain_error("Division by zero");
00246
00247     BigInt result, current;
00248     result.digits.resize(digits.size());
00249     result.negative != negative;
00250
00251     BigInt abs_this = *this; abs_this.negative = false;
00252     BigInt abs_other = other; abs_other.negative = false;
00253
00254     for (int i = static_cast<int>(digits.size()) - 1; i >= 0; --i) {
00255         current.digits.insert(current.digits.begin(), digits[i]);
00256         current.trim();
00257
00258         int x = 0, l = 0, r = 10;
00259         while (l <= r) {
00260             int m = (l + r) / 2;
00261             BigInt t = abs_other * BigInt(m);
00262             if (t <= current) {
00263                 x = m;
00264                 l = m + 1;
00265             } else {
00266                 r = m - 1;
00267             }
00268         }
00269
00270         result.digits[i] = x;
00271         current = current - abs_other * BigInt(x);
00272     }
00273
00274     result.trim();
00275     return result;
00276 }
00277
00278 BigInt BigInt::operator%(const BigInt& other) const {
00279     return *this - (*this / other) * other;
00280 }
00281
00282 // Быстрое возведение в степень по модулю
00283 BigInt BigInt::modPow(BigInt base, BigInt exp, const BigInt& mod) {
00284     base = base % mod;
00285     BigInt result(1);
00286
00287     while (!exp.isZero()) {
00288         if (exp.digits[0] % 2 == 1)
00289             result = (result * base) % mod;
00290         base = (base * base) % mod;
00291
00292         // exp = exp / 2
00293         BigInt half;
00294         half.digits.clear();
00295         int carry = 0;
00296         for (int i = static_cast<int>(exp.digits.size()) - 1; i >= 0; --i) {
00297             int current = carry * 10 + exp.digits[i];
00298             half.digits.insert(half.digits.begin(), current / 2);
00299             carry = current % 2;
00300         }
00301         half.trim();
00302         exp = half;
00303     }
00304
00305     return result;
00306 }
00307
00308 BigInt BigInt::gcd(BigInt a, BigInt b) {
00309     while (!b.isZero()) {
00310         BigInt temp = b;
00311         b = a % b;

```

```

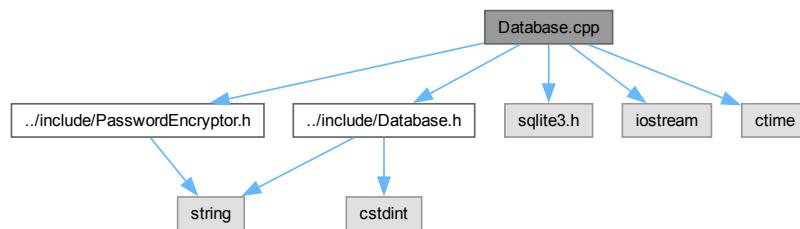
00312     a = temp;
00313 }
00314 return a;
00315 }
```

7.25 Файл Database.cpp

```

#include "../include/Database.h"
#include "../include/PasswordEncryptor.h"
#include <sqlite3.h>
#include <iostream>
#include <ctime>
```

Граф включаемых заголовочных файлов для Database.cpp:



Переменные

- sqlite3 * db = nullptr

7.25.1 Переменные

7.25.1.1 db

sqlite3* db = nullptr

См. определение в файле [Database.cpp](#) строка 7

7.26 Database.cpp

[См. документацию.](#)

```

00001 #include "../include/Database.h"
00002 #include "../include/PasswordEncryptor.h"
00003 #include <sqlite3.h>
00004 #include <iostream>
00005 #include <ctime>
00006
00007 sqlite3* db = nullptr;
00008
00009 bool Database::init(const std::string& db_path) {
00010     int rc = sqlite3_open(db_path.c_str(), &db);
00011     if (rc) {
00012         std::cerr << "Can't open database: " << sqlite3_errmsg(db) << "\n";
00013     return false;
  
```

```

00014     }
00015
00016     const char* create_users_sql = R"( 
00017         CREATE TABLE IF NOT EXISTS users (
00018             id INTEGER PRIMARY KEY AUTOINCREMENT,
00019             username TEXT UNIQUE NOT NULL,
00020             password TEXT NOT NULL
00021         );
00022     )";
00023
00024     const char* create_blacklist_sql = R"( 
00025         CREATE TABLE IF NOT EXISTS blacklist (
00026             token TEXT PRIMARY KEY,
00027             expires_at INTEGER NOT NULL
00028         );
00029     )";
00030
00031     char* errMsg = nullptr;
00032
00033     rc = sqlite3_exec(db, create_users_sql, nullptr, nullptr, &errMsg);
00034     if (rc != SQLITE_OK) {
00035         std::cerr << "SQL error (users): " << errMsg << "\n";
00036         sqlite3_free(errMsg);
00037         return false;
00038     }
00039
00040     rc = sqlite3_exec(db, create_blacklist_sql, nullptr, nullptr, &errMsg);
00041     if (rc != SQLITE_OK) {
00042         std::cerr << "SQL error (blacklist): " << errMsg << "\n";
00043         sqlite3_free(errMsg);
00044         return false;
00045     }
00046
00047     std::cout << "Database initialized successfully.\n";
00048     return true;
00049 }
00050
00051 bool Database::addUser(const std::string& username, const std::string& password) {
00052     std::string hashed = PasswordEncryptor::hashPassword(password);
00053     const char* sql = "INSERT INTO users (username, password) VALUES (?, ?);";
00054
00055     sqlite3_stmt* stmt;
00056     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00057
00058     sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_TRANSIENT);
00059     sqlite3_bind_text(stmt, 2, hashed.c_str(), -1, SQLITE_TRANSIENT);
00060
00061     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00062     sqlite3_finalize(stmt);
00063     return success;
00064 }
00065
00066 bool Database::getUser(const std::string& username, User& user_out) {
00067     const char* sql = "SELECT id, username, password FROM users WHERE username = ?;";
00068     sqlite3_stmt* stmt;
00069     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00070
00071     sqlite3_bind_text(stmt, 1, username.c_str(), -1, SQLITE_TRANSIENT);
00072
00073     bool found = false;
00074     int rc = sqlite3_step(stmt);
00075     if (rc == SQLITE_ROW) {
00076         user_out.id = sqlite3_column_int(stmt, 0);
00077         user_out.username = (const char*)sqlite3_column_text(stmt, 1);
00078         user_out.password = (const char*)sqlite3_column_text(stmt, 2);
00079         found = true;
00080     }
00081
00082     sqlite3_finalize(stmt);
00083     return found;
00084 }
00085
00086 // =====
00087 // BLACKLIST METHODS
00088 // =====
00089
00090 bool Database::blacklistToken(const std::string& token, uint64_t expires_at) {
00091     const char* sql = "INSERT OR IGNORE INTO blacklist (token, expires_at) VALUES (?, ?);";
00092     sqlite3_stmt* stmt;
00093     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00094
00095     sqlite3_bind_text(stmt, 1, token.c_str(), -1, SQLITE_TRANSIENT);
00096     sqlite3_bind_int64(stmt, 2, static_cast<sqlite3_int64>(expires_at));
00097
00098     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00099     sqlite3_finalize(stmt);
00100     return success;

```

```

00101 }
00102
00103 bool Database::isTokenBlacklisted(const std::string& token) {
00104     const char* sql = "SELECT 1 FROM blacklist WHERE token = ? LIMIT 1;";
00105     sqlite3_stmt* stmt;
00106     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00107
00108     sqlite3_bind_text(stmt, 1, token.c_str(), -1, SQLITE_TRANSIENT);
00109
00110     bool found = (sqlite3_step(stmt) == SQLITE_ROW);
00111     sqlite3_finalize(stmt);
00112     return found;
00113 }
00114
00115 bool Database::cleanupBlacklist() {
00116     const char* sql = "DELETE FROM blacklist WHERE expires_at < ?;";
00117     sqlite3_stmt* stmt;
00118     if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) return false;
00119
00120     sqlite3_bind_int64(stmt, 1, static_cast<sqlite3_int64>(std::time(nullptr)));
00121
00122     bool success = (sqlite3_step(stmt) == SQLITE_DONE);
00123     sqlite3_finalize(stmt);
00124     return success;
00125 }

```

7.27 Файл HttpServer.cpp

```

#include "../include/HttpServer.h"
#include "../include/extern/httplib.h"
#include "../include/Database.h"
#include "../include/PasswordEncryptor.h"
#include "../include/JWT.h"
#include "../include/KeyStorage.h"
#include "../include/Base64URL.h"
#include <iostream>
#include <string>

```

Граф включаемых заголовочных файлов для HttpServer.cpp:



Функции

- static std::string **extractField** (const std::string &json, const std::string &key)

7.27.1 Функции

7.27.1.1 extractField()

```

static std::string extractField (
    const std::string & json,
    const std::string & key) [static]

```

См. определение в файле [HttpServer.cpp](#) строка 12

```

00012
00013     std::string pattern = "\"" + key + "\"";
00014     size_t key_pos = json.find(pattern);
00015     if (key_pos == std::string::npos) return "";
00016

```

```

00017     size_t colon_pos = json.find(':', key_pos);
00018     if (colon_pos == std::string::npos) return "";
00019
00020     size_t quote_start = json.find('"', colon_pos + 1);
00021     if (quote_start == std::string::npos) return "";
00022
00023     size_t quote_end = json.find('"', quote_start + 1);
00024     if (quote_end == std::string::npos) return "";
00025
00026     return json.substr(quote_start + 1, quote_end - quote_start - 1);
00027 }

```

Граф вызова функции:



7.28 HttpServer.cpp

[См. документацию.](#)

```

00001 #include "../include/HttpServer.h"
00002 #include "../include/extern/httplib.h"
00003 #include "../include/Database.h"
00004 #include "../include/PasswordEncryptor.h"
00005 #include "../include/JWT.h"
00006 #include "../include/KeyStorage.h"
00007 #include "../include/Base64URL.h"
00008
00009 #include <iostream>
00010 #include <string>
00011
00012 static std::string extractField(const std::string& json, const std::string& key) {
00013     std::string pattern = "\"" + key + "\"";
00014     size_t key_pos = json.find(pattern);
00015     if (key_pos == std::string::npos) return "";
00016
00017     size_t colon_pos = json.find(':', key_pos);
00018     if (colon_pos == std::string::npos) return "";
00019
00020     size_t quote_start = json.find('"', colon_pos + 1);
00021     if (quote_start == std::string::npos) return "";
00022
00023     size_t quote_end = json.find('"', quote_start + 1);
00024     if (quote_end == std::string::npos) return "";
00025
00026     return json.substr(quote_start + 1, quote_end - quote_start - 1);
00027 }
00028
00029 void HttpServer::start(int port) {
00030     httplib::Server server;
00031
00032     server.set_logger([](const httplib::Request& req, const httplib::Response& res) {
00033         std::cout << "[LOGGER] " << req.method << " " << req.path << " -> " << res.status << "\n";
00034     });
00035
00036     server.Options(R"(.*)", [](const httplib::Request&, httplib::Response& res) {
00037         res.set_header("Access-Control-Allow-Origin", "*");
00038         res.set_header("Access-Control-Allow-Methods", "POST, GET, OPTIONS");
00039         res.set_header("Access-Control-Allow-Headers", "Content-Type, Authorization");
00040         res.status = 200;
00041     });
00042
00043     server.Post("/register", [](const httplib::Request& req, httplib::Response& res) {
00044         res.set_header("Access-Control-Allow-Origin", "*");
00045         std::cout << "[REGISTER] Получен запрос: " << req.body << std::endl;
00046
00047         std::string username = extractField(req.body, "username");

```

```

00048     std::string password = extractField(req.body, "password");
00049
00050     if (username.empty() || password.empty()) {
00051         std::cerr << "[REGISTER] Отсутствует username или password" << std::endl;
00052         res.status = 400;
00053         res.set_content("Missing 'username' or 'password'", "text/plain");
00054         return;
00055     }
00056
00057     std::cout << "[REGISTER] Имя пользователя: " << username << std::endl;
00058
00059     if (!Database::addUser(username, password)) {
00060         std::cerr << "[REGISTER] Пользователь уже существует" << std::endl;
00061         res.status = 409;
00062         res.set_content("Username already exists", "text/plain");
00063         return;
00064     }
00065
00066     std::cout << "[REGISTER] Регистрация успешна" << std::endl;
00067     res.status = 201;
00068     res.set_content("User registered successfully", "text/plain");
00069 });
00070
00071 server.Post("/login", [](const httplib::Request& req, httplib::Response& res) {
00072     res.set_header("Access-Control-Allow-Origin", "*");
00073     std::cout << "[LOGIN] Получен запрос: " << req.body << std::endl;
00074
00075     std::string username = extractField(req.body, "username");
00076     std::string password = extractField(req.body, "password");
00077
00078     if (username.empty() || password.empty()) {
00079         std::cerr << "[LOGIN] Отсутствует username или password" << std::endl;
00080         res.status = 400;
00081         res.set_content("Missing 'username' or 'password'", "text/plain");
00082         return;
00083     }
00084
00085     std::cout << "[LOGIN] Имя пользователя: " << username << std::endl;
00086
00087     User user;
00088     if (!Database::getUser(username, user)) {
00089         std::cerr << "[LOGIN] Пользователь не найден в базе данных" << std::endl;
00090         res.status = 401;
00091         res.set_content("Invalid credentials", "text/plain");
00092         return;
00093     }
00094
00095     std::string hashedInput = PasswordEncryptor::hashPassword(password);
00096     std::cout << "[LOGIN] Введённый пароль (хеш): " << hashedInput << std::endl;
00097     std::cout << "[LOGIN] Хеш пароля из базы: " << user.password << std::endl;
00098
00099     if (hashedInput != user.password) {
00100         std::cerr << "[LOGIN] Неверный пароль" << std::endl;
00101         res.status = 401;
00102         res.set_content("Invalid credentials", "text/plain");
00103         return;
00104     }
00105
00106     RSA PublicKey pubKey;
00107     RSA Private Key privKey;
00108     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00109         std::cerr << "[LOGIN] Ошибка загрузки ключей" << std::endl;
00110         res.status = 500;
00111         res.set_content("Key error", "text/plain");
00112         return;
00113     }
00114
00115     std::string accessToken = JWT::createAccessToken(username, 60 * 1, privKey); // 1 минута
00116     std::string refreshToken = JWT::createRefreshToken(username, 60 * 60, privKey); // 60 минут
00117
00118     std::cout << "[LOGIN] Сгенерирован Access токен:" << std::endl << accessToken << std::endl;
00119     std::cout << "[LOGIN] Сгенерирован Refresh токен:" << std::endl << refreshToken << std::endl;
00120
00121     std::string response = "{}";
00122     response += "\"access_token\":\"" + accessToken + "\",";
00123     response += "\"refresh_token\":\"" + refreshToken + "\",";
00124     response += "}";
00125
00126     res.set_content(response, "application/json");
00127     std::cout << "[LOGIN] Ответ отправлен клиенту\n" << std::endl;
00128 });
00129
00130 server.Post("/refresh", [](const httplib::Request& req, httplib::Response& res) {
00131     res.set_header("Access-Control-Allow-Origin", "*");
00132     std::cout << "\n[SERVER] --- /refresh endpoint called ---\n";
00133
00134     if (!req.has_header("Authorization")) {

```

```

00135     std::cerr << "[ERROR] Missing Authorization header\n";
00136     res.status = 400;
00137     res.set_content("Missing Authorization header", "text/plain");
00138     return;
00139 }
00140
00141 std::string authHeader = req.get_header_value("Authorization");
00142 std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00143
00144 std::string prefix = "Bearer ";
00145 if (authHeader.find(prefix, 0) != 0) {
00146     std::cerr << "[ERROR] Authorization header must start with 'Bearer '\n";
00147     res.status = 400;
00148     res.set_content("Invalid Authorization header format", "text/plain");
00149     return;
00150 }
00151
00152 std::string refreshToken = authHeader.substr(prefix.size());
00153 std::cout << "[PARSE] Extracted refresh token: " << refreshToken << "\n";
00154
00155 RSA PublicKey pubKey;
00156 RSA PrivateKey privKey;
00157 if (!KeyStorage::loadKeys(pubKey, privKey)) {
00158     std::cerr << "[ERROR] Failed to load RSA keys from storage\n";
00159     res.status = 500;
00160     res.set_content("Key error", "text/plain");
00161     return;
00162 }
00163
00164 std::string username;
00165 std::cout << "[VERIFY] Verifying refresh token...\n";
00166 if (!JWT::verifyRefreshToken(refreshToken, pubKey, username)) {
00167     std::cerr << "[ERROR] Invalid or expired refresh token\n";
00168     res.status = 401;
00169     res.set_content("Invalid or expired refresh token", "text/plain");
00170     return;
00171 }
00172
00173 std::cout << "[JWT] Refresh token is valid.\n";
00174 std::cout << "[JWT] Extracted subject (username): " << username << "\n";
00175
00176 if (Database::isTokenBlacklisted(refreshToken)) {
00177     std::cerr << "[SECURITY] Refresh token is blacklisted. Rejected.\n";
00178     res.status = 403;
00179     res.set_content("Refresh token is blacklisted", "text/plain");
00180     return;
00181 }
00182
00183 std::cout << "[JWT] Token is not in blacklist. Proceeding to generate new access token...\n";
00184
00185 std::string newAccessToken = JWT::createAccessToken(username, 60, privKey); // 1 минута
00186 std::cout << "[JWT] New access token generated:\n" << newAccessToken << "\n";
00187
00188 std::string response = "{";
00189 response += "access_token\":\"" + newAccessToken + "\"";
00190 response += "}";
00191
00192 std::cout << "[RESPONSE] JSON: " << response << "\n";
00193 std::cout << "[SERVER] --- /refresh complete ---\n";
00194
00195 res.set_content(response, "application/json");
00196 });
00197
00198 server.Get("/secure/data", [](const httplib::Request& req, httplib::Response& res) {
00199     res.set_header("Access-Control-Allow-Origin", "*");
00200     std::cout << "\n[SERVER] --- /secure/data endpoint called ---\n";
00201
00202     // [1] Проверяем заголовок Authorization
00203     auto authHeaderIt = req.headers.find("Authorization");
00204     if (authHeaderIt == req.headers.end()) {
00205         std::cerr << "[ERROR] Missing 'Authorization' header\n";
00206         res.status = 401;
00207         res.set_content("Missing 'Authorization' header", "text/plain");
00208         return;
00209     }
00210
00211     std::string authHeader = authHeaderIt->second;
00212     std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00213
00214     if (authHeader.find("Bearer ") != 0) {
00215         std::cerr << "[ERROR] Invalid Authorization format (should start with 'Bearer ')\n";
00216         res.status = 400;
00217         res.set_content("Invalid Authorization format", "text/plain");
00218         return;
00219     }
00220
00221     std::string accessToken = authHeader.substr(7);

```

```

00222     std::cout << "[TOKEN] Extracted access token: " << accessToken << "\n";
00223
00224     // [2] Загружаем ключи
00225     RSApublicKey pubKey;
00226     RSAprivateKey privKey; // не нужен здесь, но оставим на случай доработок
00227     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00228         std::cerr << "[ERROR] Failed to load RSA keys\n";
00229         res.status = 500;
00230         res.set_content("Key error", "text/plain");
00231         return;
00232     }
00233
00234     // [3] Проверяем токен
00235     std::string subject;
00236     std::cout << "[VERIFY] Verifying access token...\n";
00237     if (!JWT::verifyAccessToken(accessToken, pubKey, subject)) {
00238         std::cerr << "[ERROR] Invalid or expired access token\n";
00239         res.status = 401;
00240         res.set_content("Invalid or expired access token", "text/plain");
00241         return;
00242     }
00243
00244     std::cout << "[JWT] Access token is valid.\n";
00245     std::cout << "[JWT] Extracted subject (username): " << subject << "\n";
00246
00247     // [4] Возвращаем защищенные данные
00248     std::string secureData = "{ \"data\": \"Secret message for " + subject + "\" }";
00249     std::cout << "[RESPONSE] Sending secure data: " << secureData << "\n";
00250     std::cout << "[SERVER] --- /secure/data complete ---\n";
00251
00252     res.set_content(secureData, "application/json");
00253 });
00254
00255 server.Post("/logout", [](const httplib::Request& req, httplib::Response& res) {
00256     res.set_header("Access-Control-Allow-Origin", "*");
00257     std::cout << "\n[SERVER] --- /logout endpoint called ---\n";
00258
00259     if (!req.has_header("Authorization")) {
00260         std::cerr << "[ERROR] Missing Authorization header\n";
00261         res.status = 400;
00262         res.set_content("Missing Authorization header", "text/plain");
00263         return;
00264     }
00265
00266     std::string authHeader = req.get_header_value("Authorization");
00267     std::cout << "[HEADER] Authorization: " << authHeader << "\n";
00268
00269     std::string prefix = "Bearer ";
00270     if (authHeader.rfind(prefix, 0) != 0) {
00271         std::cerr << "[ERROR] Authorization header must start with 'Bearer '\n";
00272         res.status = 400;
00273         res.set_content("Invalid Authorization header format", "text/plain");
00274         return;
00275     }
00276
00277     std::string refreshToken = authHeader.substr(prefix.size());
00278     std::cout << "[INPUT] Extracted refresh token: " << refreshToken << "\n";
00279
00280     RSApublicKey pubKey;
00281     RSAprivateKey privKey;
00282     if (!KeyStorage::loadKeys(pubKey, privKey)) {
00283         std::cerr << "[ERROR] Failed to load keys\n";
00284         res.status = 500;
00285         res.set_content("Key error", "text/plain");
00286         return;
00287     }
00288
00289     std::string subject;
00290     std::cout << "[VERIFY] Verifying refresh token...\n";
00291
00292     if (!JWT::verifyRefreshToken(refreshToken, pubKey, subject)) {
00293         std::cerr << "[ERROR] Invalid or expired refresh token\n";
00294         res.status = 401;
00295         res.set_content("Invalid or expired refresh token", "text/plain");
00296         return;
00297     }
00298
00299     std::cout << "[JWT] Token is valid. Subject: " << subject << "\n";
00300
00301     // ===== Вытаскиваем expires_at из payload =====
00302     std::string payloadB64 = refreshToken.substr(
00303         refreshToken.find('.') + 1,
00304         refreshToken.rfind('.') - refreshToken.find('.') - 1
00305     );
00306     std::string payloadJson = Base64URL::decode(payloadB64);
00307
00308     size_t expPos = payloadJson.find("\\" exp \":\"");

```

```

00309     if (expPos == std::string::npos) {
00310         std::cerr << "[ERROR] Cannot extract exp from token\n";
00311         res.status = 400;
00312         res.set_content("Invalid token payload", "text/plain");
00313         return;
00314     }
00315
00316     expPos += 6;
00317     size_t expEnd = payloadJson.find_first_of(",}", expPos);
00318     uint64_t expTime = std::stoull(payloadJson.substr(expPos, expEnd - expPos));
00319
00320     std::cout << "[BLACKLIST] Extracted exp time: " << expTime << "\n";
00321
00322     if (!Database::blacklistToken(refreshToken, expTime)) {
00323         std::cerr << "[ERROR] Failed to blacklist token\n";
00324         res.status = 500;
00325         res.set_content("Database error", "text/plain");
00326         return;
00327     }
00328
00329     std::cout << "[BLACKLIST] Token successfully blacklisted\n";
00330     std::cout << "[SERVER] --- /logout completed ---\n";
00331
00332     res.set_content("Logged out successfully", "text/plain");
00333 });
00334
00335 std::cout << "[HttpServer] Сервер запущен на порту " << port << std::endl;
00336 server.listen("0.0.0.0", port);
00337 }

```

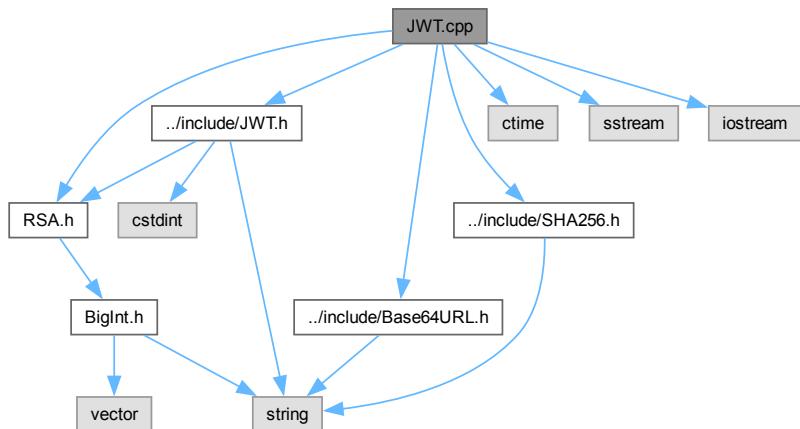
7.29 Файл JWT.cpp

```

#include "../include/JWT.h"
#include "../include/Base64URL.h"
#include "../include/SHA256.h"
#include "../include/RSA.h"
#include <ctime>
#include <sstream>
#include <iostream>

```

Граф включаемых заголовочных файлов для JWT.cpp:



7.30 JWT.cpp

[См. документацию.](#)

```

00001 #include "../include/JWT.h"
00002 #include "../include/Base64URL.h"
00003 #include "../include/SHA256.h"
00004 #include "../include/RSA.h"
00005
00006 #include <ctime>
00007 #include <sstream>
00008 #include <iostream>
00009
00010 std::string JWT::createAccessToken(const std::string& subject, uint64_t expirationSeconds, const RSAPrivateKey&
privKey) {
00011     std::string headerStr = R"({"alg":"RS256","typ":"JWT"})";
00012     std::string headerEncoded = Base64URL::encode(headerStr);
00013
00014     uint64_t now = std::time(nullptr);
00015     uint64_t exp = now + expirationSeconds;
00016
00017     std::ostringstream payloadStream;
00018     payloadStream << "{\"sub\":\"" << subject << "\",\"iat\"::" << now
00019             << ",\"exp\"::" << exp << ",\"typ\":\"access\"}";
00020
00021     std::string payloadEncoded = Base64URL::encode(payloadStream.str());
00022
00023     std::string message = headerEncoded + "." + payloadEncoded;
00024     std::string hash = SHA256::hash(message);
00025
00026     BigInt hashInt(hash, 16);
00027     BigInt signatureInt = RSA::sign(hashInt, privKey);
00028     std::string signatureStr = Base64URL::encode(signatureInt.toString(16));
00029
00030     std::string token = message + "." + signatureStr;
00031
00032     std::cout << "[JWT::createAccessToken] ---" << std::endl;
00033     std::cout << "Header JSON: " << headerStr << std::endl;
00034     std::cout << "Payload JSON: " << payloadStream.str() << std::endl;
00035     std::cout << "Header Encoded: " << headerEncoded << std::endl;
00036     std::cout << "Payload Encoded: " << payloadEncoded << std::endl;
00037     std::cout << "Message (header.payload): " << message << std::endl;
00038     std::cout << "SHA256 Hash: " << hash << std::endl;
00039     std::cout << "Signature (hex): " << signatureInt.toString(16) << std::endl;
00040     std::cout << "Access Token: " << token << std::endl;
00041
00042     return token;
00043 }
00044
00045 std::string JWT::createRefreshToken(const std::string& subject, uint64_t expirationSeconds, const RSAPrivateKey&
privKey) {
00046     std::string headerStr = R"({"alg":"RS256","typ":"JWT"})";
00047     std::string headerEncoded = Base64URL::encode(headerStr);
00048
00049     uint64_t now = std::time(nullptr);
00050     uint64_t exp = now + expirationSeconds;
00051
00052     std::ostringstream payloadStream;
00053     payloadStream << "{\"sub\":\"" << subject << "\",\"iat\"::" << now
00054             << ",\"exp\"::" << exp << ",\"typ\":\"refresh\"}";
00055
00056     std::string payloadEncoded = Base64URL::encode(payloadStream.str());
00057
00058     std::string message = headerEncoded + "." + payloadEncoded;
00059     std::string hash = SHA256::hash(message);
00060
00061     BigInt hashInt(hash, 16);
00062     BigInt signatureInt = RSA::sign(hashInt, privKey);
00063     std::string signatureStr = Base64URL::encode(signatureInt.toString(16));
00064
00065     std::string token = message + "." + signatureStr;
00066
00067     std::cout << "[JWT::createRefreshToken] ---" << std::endl;
00068     std::cout << "Header JSON: " << headerStr << std::endl;
00069     std::cout << "Payload JSON: " << payloadStream.str() << std::endl;
00070     std::cout << "Header Encoded: " << headerEncoded << std::endl;
00071     std::cout << "Payload Encoded: " << payloadEncoded << std::endl;
00072     std::cout << "Message (header.payload): " << message << std::endl;
00073     std::cout << "SHA256 Hash: " << hash << std::endl;
00074     std::cout << "Signature (hex): " << signatureInt.toString(16) << std::endl;
00075     std::cout << "Refresh Token: " << token << std::endl;
00076
00077     return token;
00078 }
00079
00080 bool JWT::verifyAccessToken(const std::string& token, const RSAPublicKey& pubKey, std::string& outSubject) {
00081     std::cout << "[JWT::verifyAccessToken] ---" << std::endl;
00082     std::cout << "Received token: " << token << std::endl;
00083
00084     size_t firstDot = token.find('.');
00085     size_t secondDot = token.find('.', firstDot + 1);

```

```

00086     if (firstDot == std::string::npos || secondDot == std::string::npos) return false;
00087
00088     std::string headerB64 = token.substr(0, firstDot);
00089     std::string payloadB64 = token.substr(firstDot + 1, secondDot - firstDot - 1);
00090     std::string signatureB64 = token.substr(secondDot + 1);
00091
00092     std::string message = headerB64 + "." + payloadB64;
00093     std::string expectedHash = SHA256::hash(message);
00094
00095     std::string sigHex = Base64URL::decode(signatureB64);
00096     BigInt signature(sigHex, 16);
00097
00098     if (!RSA::verify(expectedHash, signature, pubKey)) {
00099         std::cerr << "[JWT] Подпись access токена недействительна" << std::endl;
00100         return false;
00101     }
00102
00103     std::string payloadJson = Base64URL::decode(payloadB64);
00104     std::cout << "Decoded payload: " << payloadJson << std::endl;
00105
00106     if (payloadJson.find("\\"typ\":\\access\\") == std::string::npos) {
00107         std::cerr << "[JWT] Токен не является access" << std::endl;
00108         return false;
00109     }
00110
00111     size_t subPos = payloadJson.find("\\sub\\:");
00112     size_t expPos = payloadJson.find("\\exp\\:");
00113
00114     if (subPos == std::string::npos || expPos == std::string::npos) return false;
00115
00116     subPos += 7;
00117     size_t subEnd = payloadJson.find("\\", subPos);
00118     outSubject = payloadJson.substr(subPos, subEnd - subPos);
00119
00120     expPos += 6;
00121     size_t expEnd = payloadJson.find_first_of(",}", expPos);
00122     std::string expStr = payloadJson.substr(expPos, expEnd - expPos);
00123     uint64_t exp = std::stoull(expStr);
00124
00125     std::cout << "Subject: " << outSubject << std::endl;
00126     std::cout << "Expiration: " << exp << ", now: " << std::time(nullptr) << std::endl;
00127
00128     if (static_cast<uint64_t>(std::time(nullptr)) > exp) {
00129         std::cerr << "[JWT] Access токен просрочен" << std::endl;
00130         return false;
00131     }
00132
00133     return true;
00134 }
00135
00136 bool JWT::verifyRefreshToken(const std::string& token, const RSAPublicKey& pubKey, std::string& outSubject) {
00137     std::cout << "[JWT::verifyRefreshToken] ---" << std::endl;
00138     std::cout << "Received token: " << token << std::endl;
00139
00140     size_t firstDot = token.find('.');
00141     size_t secondDot = token.find('.', firstDot + 1);
00142     if (firstDot == std::string::npos || secondDot == std::string::npos) return false;
00143
00144     std::string headerB64 = token.substr(0, firstDot);
00145     std::string payloadB64 = token.substr(firstDot + 1, secondDot - firstDot - 1);
00146     std::string signatureB64 = token.substr(secondDot + 1);
00147
00148     std::string message = headerB64 + "." + payloadB64;
00149     std::string expectedHash = SHA256::hash(message);
00150
00151     std::string sigHex = Base64URL::decode(signatureB64);
00152     BigInt signature(sigHex, 16);
00153
00154     if (!RSA::verify(expectedHash, signature, pubKey)) {
00155         std::cerr << "[JWT] Refresh подпись недействительна" << std::endl;
00156         return false;
00157     }
00158
00159     std::string payloadJson = Base64URL::decode(payloadB64);
00160     std::cout << "Decoded payload: " << payloadJson << std::endl;
00161
00162     if (payloadJson.find("\\"typ\":\\refresh\\") == std::string::npos) {
00163         std::cerr << "[JWT] Токен не является refresh" << std::endl;
00164         return false;
00165     }
00166
00167     size_t subPos = payloadJson.find("\\sub\\:");
00168     size_t expPos = payloadJson.find("\\exp\\:");
00169     if (subPos == std::string::npos || expPos == std::string::npos) return false;
00170
00171     subPos += 7;
00172     size_t subEnd = payloadJson.find("\\", subPos);

```

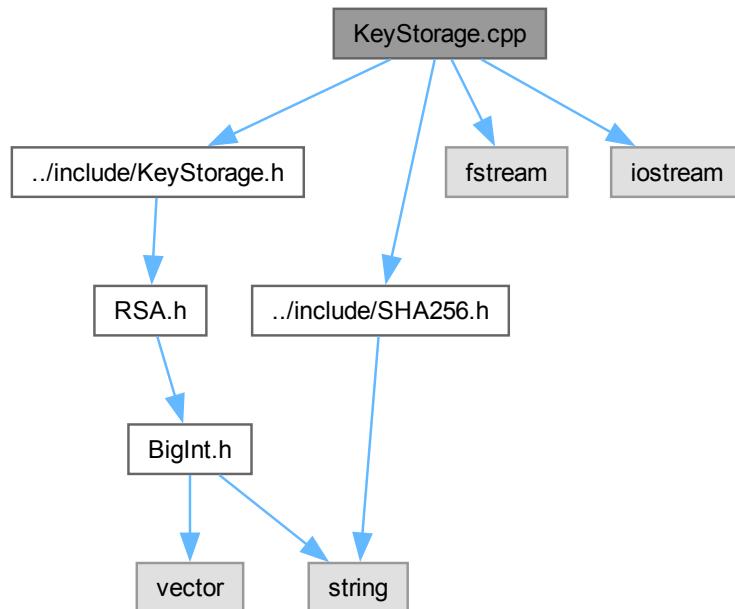
```

00173     outSubject = payloadJson.substr(subPos, subEnd - subPos);
00174
00175     expPos += 6;
00176     size_t expEnd = payloadJson.find_first_of("},{", expPos);
00177     std::string expStr = payloadJson.substr(expPos, expEnd - expPos);
00178     uint64_t exp = std::stoull(expStr);
00179
00180     std::cout << "Subject: " << outSubject << std::endl;
00181     std::cout << "Expiration: " << exp << ", now: " << std::time(nullptr) << std::endl;
00182
00183     if (static_cast<uint64_t>(std::time(nullptr)) > exp) {
00184         std::cerr << "[JWT] Refresh токен просрочен" << std::endl;
00185         return false;
00186     }
00187
00188     return true;
00189 }
```

7.31 Файл KeyStorage.cpp

```
#include "../include/KeyStorage.h"
#include "../include/SHA256.h"
#include <fstream>
#include <iostream>
```

Граф включаемых заголовочных файлов для KeyStorage.cpp:



Переменные

- static const std::string **PRIV_FILE** = "rsa_private.key"
- static const std::string **PUB_FILE** = "rsa_public.key"

7.31.1 Переменные

7.31.1.1 PRIV_FILE

```
const std::string PRIV_FILE = "rsa_private.key" [static]
```

См. определение в файле [KeyStorage.cpp](#) строка 6

7.31.1.2 PUB_FILE

```
const std::string PUB_FILE = "rsa_public.key" [static]
```

См. определение в файле [KeyStorage.cpp](#) строка 7

7.32 KeyStorage.cpp

[См. документацию.](#)

```
00001 #include "../include/KeyStorage.h"
00002 #include "../include/SHA256.h"
00003 #include <fstream>
00004 #include <iostream>
00005
00006 static const std::string PRIV_FILE = "rsa_private.key";
00007 static const std::string PUB_FILE = "rsa_public.key";
00008
00009 void KeyStorage::saveKeys(const RSApublicKey& pubKey, const RSAprivateKey& privKey) {
00010     // Save private key with SHA256 hash
00011     std::ofstream privOut(PRIV_FILE);
00012     if (privOut) {
00013         std::string content = privKey.d.toString() + ";" + privKey.n.toString();
00014         std::string hash = SHA256::hash(content);
00015         privOut << content << "\n" << "hash=" << hash << "\n";
00016     }
00017
00018     // Save public key
00019     std::ofstream pubOut(PUB_FILE);
00020     if (pubOut) {
00021         pubOut << pubKey.e.toString() << ";" << pubKey.n.toString() << "\n";
00022     }
00023 }
00024
00025 bool KeyStorage::loadKeys(RSApublicKey& pubKey, RSAprivateKey& privKey) {
00026     std::ifstream privIn(PRIV_FILE);
00027     std::ifstream pubIn(PUB_FILE);
00028
00029     if (!privIn || !pubIn) return false;
00030
00031     std::string privLine, hashLine;
00032     if (!std::getline(privIn, privLine) || !std::getline(privIn, hashLine)) return false;
00033
00034     // validate hash
00035     std::string expectedHash = SHA256::hash(privLine);
00036     if (hashLine != "hash=" + expectedHash) {
00037         std::cerr << "[KeyStorage] Ошибка: контрольная сумма не совпадает. Приватный ключ поврежден." << std::endl;
00038         return false;
00039     }
00040
00041     size_t delimPos = privLine.find(';');
00042     if (delimPos == std::string::npos) return false;
00043     std::string dStr = privLine.substr(0, delimPos);
00044     std::string nStr = privLine.substr(delimPos + 1);
00045
00046     privKey.d = BigInt(dStr);
00047     privKey.n = BigInt(nStr);
00048
00049     // Load public key
00050     std::string pubLine;
00051     if (!std::getline(pubIn, pubLine)) return false;
00052
00053     delimPos = pubLine.find(';');
00054     if (delimPos == std::string::npos) return false;
```

```

00055     std::string eStr = pubLine.substr(0, delimPos);
00056     std::string pubNStr = pubLine.substr(delimPos + 1);
00057
00058     pubKey.e = BigInt(eStr);
00059     pubKey.n = BigInt(pubNStr);
00060
00061     return true;
00062 }
```

7.33 Файл main.cpp

```
#include "../include/HttpServer.h"
#include "../include/Database.h"
#include "../include/KeyStorage.h"
```

Граф включаемых заголовочных файлов для main.cpp:



Функции

- int **main ()**

7.33.1 Функции

7.33.1.1 main()

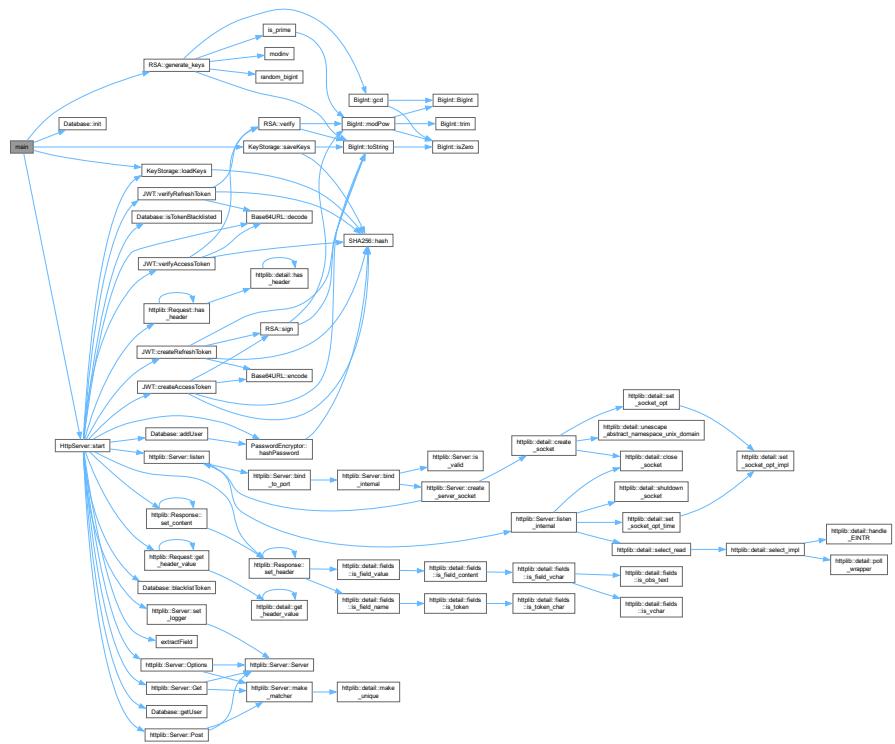
```
int main ()
```

См. определение в файле **main.cpp** строка 5

```

00005     {
00006     Database::init("users.db");
00007
00008     RSAPublicKey pubKey;
00009     RSAPrivateKey privKey;
0010  if (!KeyStorage::loadKeys(pubKey, privKey)) {
0011      std::cout << "[main] Ключи не найдены, создаю заново...\n";
0012      RSA::generate_keys(pubKey, privKey, 256);
0013      KeyStorage::saveKeys(pubKey, privKey);
0014  }
0015
0016  HttpServer::start(8080);
0017  return 0;
0018 }
```

Граф вызовов:



7.34 main.cpp

[См. документацию.](#)

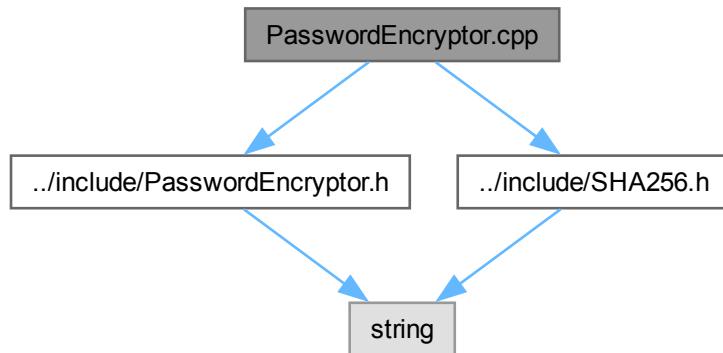
```

00001 #include "../include/HttpServer.h"
00002 #include "../include/Database.h"
00003 #include "../include/KeyStorage.h"
00004
00005 int main() {
00006     Database::init("users.db");
00007
00008     RSA PublicKey;
00009     RSA PrivateKey;
0010     if (!KeyStorage::loadKeys(pubKey, privKey)) {
0011         std::cout << "[main] Ключи не найдены, создаю заново...\n";
0012         RSA::generate_keys(pubKey, privKey, 256);
0013         KeyStorage::saveKeys(pubKey, privKey);
0014     }
0015
0016     HttpServer::start(8080);
0017     return 0;
0018 }
```

7.35 Файл PasswordEncryptor.cpp

```
#include "../include/PasswordEncryptor.h"
#include "../include/SHA256.h"
```

Граф включаемых заголовочных файлов для PasswordEncryptor.cpp:



7.36 PasswordEncryptor.cpp

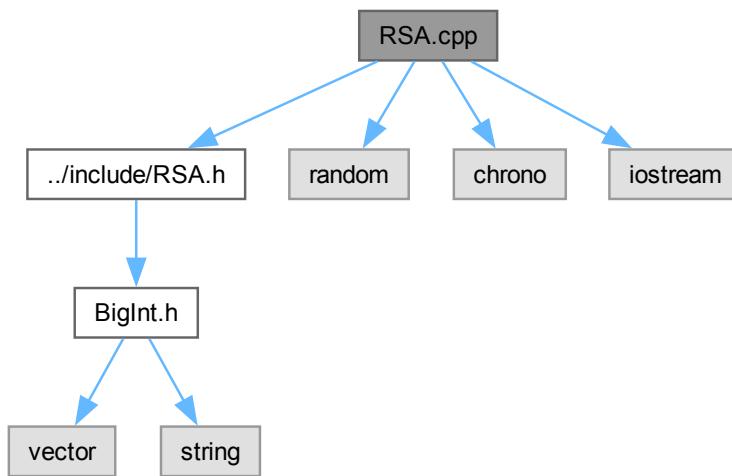
[См. документацию.](#)

```
00001 #include "../include/PasswordEncryptor.h"
00002 #include "../include/SHA256.h"
00003
00004 std::string PasswordEncryptor::hashPassword(const std::string& password) {
00005     return SHA256::hash(password);
00006 }
00007
```

7.37 Файл RSA.cpp

```
#include "../include/RSA.h"
#include <random>
#include <chrono>
#include <iostream>
```

Граф включаемых заголовочных файлов для RSA.cpp:



Функции

- static bool `is_prime` (const `BigInt` &n, int iterations=10)
- static `BigInt random bigint` (int bit_length)
- static `BigInt modinv` (const `BigInt` &a, const `BigInt` &m)

7.37.1 Функции

7.37.1.1 `is_prime()`

```
static bool is_prime (
    const BigInt & n,
    int iterations = 10) [static]
```

См. определение в файле RSA.cpp строка 6

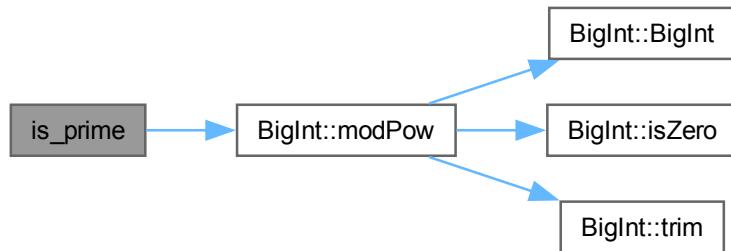
```

00006
00007     if (n <= BigInt(1)) return false;
00008     if (n == BigInt(2) || n == BigInt(3)) return true;
00009     if (n % BigInt(2) == BigInt(0)) return false;
00010
00011     BigInt d = n - BigInt(1);
00012     int r = 0;
00013
00014     while (d % BigInt(2) == BigInt(0)) {
00015         d = d / BigInt(2);
00016         r++;
00017     }
00018
00019     std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
00020
00021     for (int i = 0; i < iterations; ++i) {
00022         BigInt a = BigInt(2 + rng() % 10000);
00023         BigInt x = BigInt::modPow(a, d, n);
00024         if (x == BigInt(1) || x == n - BigInt(1)) continue;
00025
00026         bool continue_outer = false;
00027         for (int j = 0; j < r - 1; ++j) {
```

```

00028     x = BigInt::modPow(x, BigInt(2), n);
00029     if (x == n - BigInt(1)) {
00030         continue_outer = true;
00031         break;
00032     }
00033 }
00034
00035     if (continue_outer) continue;
00036     return false;
00037 }
00038
00039 return true;
00040 }
```

Граф вызовов:



Граф вызова функции:



7.37.1.2 modinv()

```

static BigInt modinv (
    const BigInt & a,
    const BigInt & m) [static]
```

См. определение в файле RSA.cpp строка 52

```

00052                                     {
00053     BigInt m0 = m, t, q;
00054     BigInt x0 = 0, x1 = 1;
00055     BigInt a_ = a;
00056     BigInt m_ = m;
00057
00058     while (a_ > 1) {
00059         q = a_ / m_;
00060         t = m_;
00061         m_ = a_ % m_, a_ = t;
00062         t = x0;
```

```

00063     x0 = x1 - q * x0;
00064     x1 = t;
00065 }
00066
00067 if (x1 < 0)
00068     x1 = x1 + m0;
00069
00070 return x1;
00071 }
```

Граф вызова функции:



7.37.1.3 random bigint()

```

static BigInt random bigint (
    int bit_length) [static]
```

См. определение в файле RSA.cpp строка 42

```

00042                         {
00043     std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
00044     BigInt result(0);
00045     for (int i = 0; i < bit_length; i += 4) {
00046         result = result * BigInt(10);
00047         result = result + BigInt(rng() % 10);
00048     }
00049     return result;
00050 }
```

Граф вызова функции:



7.38 RSA.cpp

[См. документацию.](#)

```

00001 #include "../include/RSA.h"
00002 #include <random>
00003 #include <chrono>
00004 #include <iostream>
00005
00006 static bool is_prime(const BigInt& n, int iterations = 10) {
00007     if (n <= BigInt(1)) return false;
00008     if (n == BigInt(2) || n == BigInt(3)) return true;
```

```

00009 if (n % BigInt(2) == BigInt(0)) return false;
00010
00011 BigInt d = n - BigInt(1);
00012 int r = 0;
00013
00014 while (d % BigInt(2) == BigInt(0)) {
00015     d = d / BigInt(2);
00016     r++;
00017 }
00018
00019 std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
00020
00021 for (int i = 0; i < iterations; ++i) {
00022     BigInt a = BigInt(2 + rng() % 10000);
00023     BigInt x = BigInt::modPow(a, d, n);
00024     if (x == BigInt(1) || x == n - BigInt(1)) continue;
00025
00026     bool continue_outer = false;
00027     for (int j = 0; j < r - 1; ++j) {
00028         x = BigInt::modPow(x, BigInt(2), n);
00029         if (x == n - BigInt(1)) {
00030             continue_outer = true;
00031             break;
00032         }
00033     }
00034
00035     if (continue_outer) continue;
00036     return false;
00037 }
00038
00039 return true;
00040 }
00041
00042 static BigInt random bigint(int bit_length) {
00043     std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
00044     BigInt result(0);
00045     for (int i = 0; i < bit_length; i += 4) {
00046         result = result * BigInt(10);
00047         result = result + BigInt(rng() % 10);
00048     }
00049     return result;
00050 }
00051
00052 static BigInt modinv(const BigInt& a, const BigInt& m) {
00053     BigInt m0 = m, t, q;
00054     BigInt x0 = 0, x1 = 1;
00055     BigInt a_ = a;
00056     BigInt m_ = m;
00057
00058     while (a_ > 1) {
00059         q = a_ / m_;
00060         t = m_;
00061         m_ = a_ % m_, a_ = t;
00062         t = x0;
00063         x0 = x1 - q * x0;
00064         x1 = t;
00065     }
00066
00067     if (x1 < 0)
00068         x1 = x1 + m0;
00069
00070     return x1;
00071 }
00072
00073 void RSA::generate_keys(RSAPublicKey& pub, RSAPrivatekey& priv, int bit_length) {
00074     std::cout << "[RSA] --- Генерация ключей ---" << std::endl;
00075
00076     BigInt p, q;
00077
00078     std::cout << "[RSA] Генерация простого p..." << std::endl;
00079     do {
00080         p = random bigint(bit_length);
00081     } while (!is_prime(p));
00082     std::cout << "[RSA] Простое p: " << p.toString() << std::endl;
00083
00084     std::cout << "[RSA] Генерация простого q..." << std::endl;
00085     do {
00086         q = random bigint(bit_length);
00087     } while (!is_prime(q) || p == q);
00088     std::cout << "[RSA] Простое q: " << q.toString() << std::endl;
00089
00090     BigInt n = p * q;
00091     BigInt phi = (p - BigInt(1)) * (q - BigInt(1));
00092     BigInt e = 65537;
00093
00094     std::cout << "[RSA] n = p * q = " << n.toString() << std::endl;
00095     std::cout << "[RSA] phi = (p-1)*(q-1) = " << phi.toString() << std::endl;

```

```

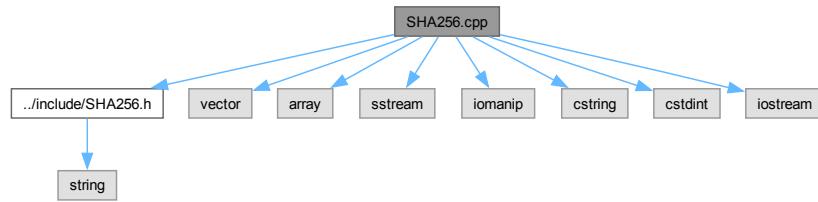
00096
00097     while (BigInt::gcd(e, phi) != BigInt(1)) {
00098         e = e + BigInt(2);
00099     }
00100
00101     std::cout << "[RSA] Публичная экспонента e: " << e.toString() << std::endl;
00102
00103     BigInt d = modinv(e, phi);
00104     std::cout << "[RSA] Приватная экспонента d: " << d.toString() << std::endl;
00105
00106     pub = {e, n};
00107     priv = {d, n};
00108
00109     std::cout << "[RSA] --- Ключи успешно сгенерированы ---" << std::endl;
00110 }
00111
00112 BigInt RSA::encrypt(const BigInt& message, const RSAPublicKey& key) {
00113     std::cout << "[RSA] --- Шифрование ---" << std::endl;
00114     std::cout << "Message: " << message.toString(16) << std::endl;
00115     BigInt cipher = BigInt::modPow(message, key.e, key.n);
00116     std::cout << "Encrypted: " << cipher.toString(16) << std::endl;
00117     return cipher;
00118 }
00119
00120 BigInt RSA::decrypt(const BigInt& cipher, const RSAPrivateKey& key) {
00121     std::cout << "[RSA] --- Расшифровка ---" << std::endl;
00122     std::cout << "Cipher: " << cipher.toString(16) << std::endl;
00123     BigInt message = BigInt::modPow(cipher, key.d, key.n);
00124     std::cout << "Decrypted: " << message.toString(16) << std::endl;
00125     return message;
00126 }
00127
00128 BigInt RSA::sign(const BigInt& hash, const RSAPrivateKey& key) {
00129     std::cout << "[RSA] --- Подпись ---" << std::endl;
00130     std::cout << "Hash (hex): " << hash.toString(16) << std::endl;
00131     BigInt sig = BigInt::modPow(hash, key.d, key.n);
00132     std::cout << "Signature (hex): " << sig.toString(16) << std::endl;
00133     return sig;
00134 }
00135
00136 bool RSA::verify(const std::string& messageHashHex, const BigInt& signature, const RSAPublicKey& key) {
00137     std::cout << "[RSA] --- Верификация подписи ---" << std::endl;
00138     std::cout << "Expected hash: " << messageHashHex << std::endl;
00139     std::cout << "Signature: " << signature.toString(16) << std::endl;
00140
00141     BigInt decryptedHashInt = BigInt::modPow(signature, key.e, key.n);
00142     std::string decryptedHashHex = decryptedHashInt.toString(16);
00143
00144     while (decryptedHashHex.length() < 64)
00145         decryptedHashHex = "0" + decryptedHashHex;
00146
00147     std::cout << "Decrypted hash: " << decryptedHashHex << std::endl;
00148
00149     bool valid = (decryptedHashHex == messageHashHex);
00150     std::cout << "Signature valid: " << (valid ? "YES" : "NO") << std::endl;
00151
00152     return valid;
00153 }
```

7.39 Файл SHA256.cpp

```

#include "../include/SHA256.h"
#include <vector>
#include <array>
#include <sstream>
#include <iomanip>
#include <cstring>
#include <cstdint>
#include <iostream>
```

Граф включаемых заголовочных файлов для SHA256.cpp:



Пространства имен

- namespace [anonymous_namespace{SHA256.cpp}](#)

Функции

- uint32_t [anonymous_namespace{SHA256.cpp}::rotr](#) (uint32_t x, uint32_t n)
Побитовый циклический сдвиг вправо.
- uint32_t [anonymous_namespace{SHA256.cpp}::ch](#) (uint32_t x, uint32_t y, uint32_t z)
Вычисляет функцию выбора: выбирает y, если x, иначе z.
- uint32_t [anonymous_namespace{SHA256.cpp}::maj](#) (uint32_t x, uint32_t y, uint32_t z)
Мажоритарная функция: возвращает значение, встречающееся чаще всего среди x, y, z.
- uint32_t [anonymous_namespace{SHA256.cpp}::big_sigma0](#) (uint32_t x)
Функция 0 из SHA-256: используется в расширении блока.
- uint32_t [anonymous_namespace{SHA256.cpp}::big_sigma1](#) (uint32_t x)
Функция 1 из SHA-256: используется в расширении блока.
- uint32_t [anonymous_namespace{SHA256.cpp}::small_sigma0](#) (uint32_t x)
Функция 0 из SHA-256: используется при генерации w[16..63].
- uint32_t [anonymous_namespace{SHA256.cpp}::small_sigma1](#) (uint32_t x)
Функция 1 из SHA-256: используется при генерации w[16..63].
- std::vector< uint8_t > [anonymous_namespace{SHA256.cpp}::pad](#) (const std::string &input)
Дополняет вход до кратного 512 бит (64 байта).
- uint32_t [anonymous_namespace{SHA256.cpp}::to_uint32](#) (const uint8_t *bytes)
Преобразует 4 байта в 32-битное целое число (big-endian).

Переменные

- const std::array< uint32_t, 64 > [anonymous_namespace{SHA256.cpp}::k](#)
Константы, используемые в каждом из 64 раундов SHA-256.

7.40 SHA256.cpp

[См. документацию.](#)

```

00001 #include "../include/SHA256.h"
00002 #include <vector>
00003 #include <array>
00004 #include <iomanip>
00005 #include <iomanip>
00006 #include <cstring>
00007 #include <cstdint>
00008 #include <iostream>
00009
0010 namespace {
0016   const std::array<uint32_t, 64> k = {
0017     0x428a2f98, 0x71374491, 0xb5c0fbef, 0xe9b5dba5,
0018     0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0019     0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
0020     0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
0021     0xe49b69c1, 0xefbe4786, 0xfc19dc6, 0x240ca1cc,
0022     0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0023     0x983e5152, 0xa831c66d, 0xb00327c8, 0xb5f597fc7,
0024     0xc6e00bf3, 0xd5a79147, 0x6ca6351, 0x14292967,
0025     0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
0026     0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0027     0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0x76c51a3,
0028     0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0029     0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
0030     0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0031     0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0032     0x90beffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
0033   };
0034
0038 inline uint32_t rotr(uint32_t x, uint32_t n) {
0039   return (x >> n) | (x << (32 - n));
0040 }
0041
0047 inline uint32_t ch(uint32_t x, uint32_t y, uint32_t z) {
0048   return (x & y) ^ (~x & z);
0049 }
0050
0056 inline uint32_t maj(uint32_t x, uint32_t y, uint32_t z) {
0057   return (x & y) ^ (x & z) ^ (y & z);
0058 }
0059
0063 inline uint32_t big_sigma0(uint32_t x) {
0064   return rotr(x, 2) ^ rotr(x, 13) ^ rotr(x, 22);
0065 }
0066
0070 inline uint32_t big_sigma1(uint32_t x) {
0071   return rotr(x, 6) ^ rotr(x, 11) ^ rotr(x, 25);
0072 }
0073
0077 inline uint32_t small_sigma0(uint32_t x) {
0078   return rotr(x, 7) ^ rotr(x, 18) ^ (x >> 3);
0079 }
0080
0084 inline uint32_t small_sigma1(uint32_t x) {
0085   return rotr(x, 17) ^ rotr(x, 19) ^ (x >> 10);
0086 }
0087
0099 std::vector<uint8_t> pad(const std::string& input) {
00100   size_t original_length = input.size();
00101   uint64_t bit_length = original_length * 8;
00102
00103   std::vector<uint8_t> padded(input.begin(), input.end());
00104   padded.push_back(0x80);
00105   while ((padded.size() + 8) % 64 != 0) padded.push_back(0x00);
00106   for (int i = 7; i >= 0; --i)
00107     padded.push_back((bit_length >> (i * 8)) & 0xFF);
00108
00109   return padded;
00110 }
00111
00115 uint32_t to_uint32(const uint8_t* bytes) {
00116   return (bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3];
00117 }
00118 }
00119
00120 std::string SHA256::hash(const std::string& input) {
00121   std::cout << "[SHA256] --- Начало хеширования ---" << std::endl;
00122   std::cout << "[SHA256] Входная строка: " << input << std::endl;
00123
00124   std::array<uint32_t, 8> h = {
00125     0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
00126     0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19

```

```
00127    };
00128
00129    std::vector<uint8_t> data = pad(input);
00130    std::cout << "[SHA256] Размер после паддинга: " << data.size() << " байт" << std::endl;
00131
00132    for (size_t i = 0; i < data.size(); i += 64) {
00133        uint32_t w[64];
00134
00135        for (int j = 0; j < 16; ++j)
00136            w[j] = to_uint32(&data[i + j * 4]);
00137
00138        for (int j = 16; j < 64; ++j)
00139            w[j] = small_sigma1(w[j - 2]) + w[j - 7] +
00140                small_sigma0(w[j - 15]) + w[j - 16];
00141
00142        uint32_t a = h[0], b = h[1], c = h[2], d = h[3];
00143        uint32_t e = h[4], f = h[5], g = h[6], h_val = h[7];
00144
00145        std::cout << "[SHA256] Обработка блока #" << i / 64 << std::endl;
00146
00147        for (int j = 0; j < 64; ++j) {
00148            uint32_t temp1 = h_val + big_sigma1(e) + ch(e, f, g) + k[j] + w[j];
00149            uint32_t temp2 = big_sigma0(a) + maj(a, b, c);
00150
00151            h_val = g;
00152            g = f;
00153            f = e;
00154            e = d + temp1;
00155            d = c;
00156            c = b;
00157            b = a;
00158            a = temp1 + temp2;
00159
00160            if (j < 4 || j > 59) { // лог только в начале и в конце
00161                std::cout << "Round " << j << ": a=" << std::hex << a << " e=" << e << std::dec << std::endl;
00162            }
00163        }
00164
00165        h[0] += a; h[1] += b; h[2] += c; h[3] += d;
00166        h[4] += e; h[5] += f; h[6] += g; h[7] += h_val;
00167    }
00168
00169    std::ostringstream oss;
00170    for (auto val : h) {
00171        oss << std::hex << std::setw(8) << std::setfill('0') << val;
00172    }
00173
00174    std::string result = oss.str();
00175    std::cout << "[SHA256] Итоговый хеш: " << result << std::endl;
00176    std::cout << "[SHA256] --- Завершено ---" << std::endl;
00177
00178    return result;
00179 }
```

Предметный указатель

~BufferStream
 `httpplib::detail::BufferStream`, 163

~Client
 `httpplib::Client`, 173

~ClientImpl
 `httpplib::ClientImpl`, 208

~MatcherBase
 `httpplib::detail::MatcherBase`, 349

~Response
 `httpplib::Response`, 399

~Server
 `httpplib::Server`, 438

~SocketStream
 `httpplib::detail::SocketStream`, 502

~Stream
 `httpplib::Stream`, 511

~TaskQueue
 `httpplib::TaskQueue`, 522

~ThreadPool
 `httpplib::ThreadPool`, 526

~compressor
 `httpplib::detail::compressor`, 306

~decompressor
 `httpplib::detail::decompressor`, 325

~mmap
 `httpplib::detail::mmap`, 351

~nocompressor
 `httpplib::detail::nocompressor`, 372

~scope_exit
 `httpplib::detail::scope_exit`, 429

Accepted_202
 `httpplib`, 18

addr_
 `httpplib::detail::mmap`, 355

addr_map_
 `httpplib::ClientImpl`, 300

address_family_
 `httpplib::ClientImpl`, 300
 `httpplib::Server`, 489

addUser
 `Database`, 315

adjust_host_string
 `httpplib::ClientImpl`, 209

AlreadyReported_208
 `httpplib`, 18

anonymous_namespace{SHA256.cpp}, 7
 `big_sigma0`, 8
 `big_sigma1`, 8

`ch`, 8
 `k`, 12
 `maj`, 9
 `pad`, 9
 `rotr`, 9
 `small_sigma0`, 10
 `small_sigma1`, 11
 `to_uint32`, 11

append
 `httpplib::detail::stream_line_reader`, 517

append_query_params
 `httpplib`, 20

apply_ranges
 `httpplib::Server`, 439

authorization_count_
 `httpplib::Request`, 391

BadGateway_502
 `httpplib`, 19

BadRequest_400
 `httpplib`, 18

base64_chars
 `Base64URL.cpp`, 677

base64_encode
 `httpplib::detail`, 32

Base64URL, 136
 `decode`, 137
 `encode`, 138

Base64URL.cpp, 677
 `base64_chars`, 677

Base64URL.h, 533, 534

base_dir
 `httpplib::Server::MountPointEntry`, 356

base_dirs_
 `httpplib::Server`, 489

basic_auth_password
 `httpplib::ClientImpl`, 300

basic_auth_username_
 `httpplib::ClientImpl`, 300

bearer_token_auth_token_
 `httpplib::ClientImpl`, 300

big_sigma0
 `anonymous_namespace{SHA256.cpp}`, 8

big_sigma1
 `anonymous_namespace{SHA256.cpp}`, 8

BigInt, 140
 `BigInt`, 142, 143
 `compareAbs`, 145
 `digits`, 159

gcd, 145
isNegative, 146
isZero, 147
modPow, 147
negative, 159
operator!=, 149
operator<, 154
operator<=, 155
operator>, 156
operator>=, 156
operator+, 151
operator-, 152
operator/, 153
operator==, 155
operator%, 149
operator*, 150
toString, 157, 158
trim, 159

BigInt.cpp, 678, 679
BigInt.h, 534, 535
bind_internal
 httpplib::Server, 441
bind_ip_address
 httpplib::detail, 32
bind_to_any_port
 httpplib::Server, 442
bind_to_port
 httpplib::Server, 442
BindIPAddress
 httpplib, 17
blacklistToken
 Database, 315
body
 httpplib::Request, 391
 httpplib::Response, 409
boundary_
 httpplib::detail::MultipartFormDataParser, 368
Brotli
 httpplib::detail, 32
buf_
 httpplib::detail::MultipartFormDataParser, 368
buf_append
 httpplib::detail::MultipartFormDataParser, 360
buf_data
 httpplib::detail::MultipartFormDataParser, 360
buf_epos_
 httpplib::detail::MultipartFormDataParser, 368
buf_erase
 httpplib::detail::MultipartFormDataParser, 360
buf_find
 httpplib::detail::MultipartFormDataParser, 361
buf_head
 httpplib::detail::MultipartFormDataParser, 362
buf_size
 httpplib::detail::MultipartFormDataParser, 362
buf_spos_
 httpplib::detail::MultipartFormDataParser, 368
buf_start_with

 httpplib::detail::MultipartFormDataParser, 362
buffer
 httpplib::detail::BufferStream, 166
BufferStream
 httpplib::detail::BufferStream, 163
calc_actual_timeout
 httpplib::detail, 33
Callback
 httpplib::detail::compressor, 306
 httpplib::detail::decompressor, 325
can_compress_content_type
 httpplib::detail, 34
Canceled
 httpplib, 17
CertificateAccepted
 httpplib, 17
CertificateRejected
 httpplib, 17
ch
 anonymous_namespace{SHA256.cpp}, 8
cleanupBlacklist
 Database, 316
clear_file_info
 httpplib::detail::MultipartFormDataParser, 363
cli_
 httpplib::Client, 200
Client
 httpplib::Client, 170–173
client_cert_path_
 httpplib::ClientImpl, 300
client_key_path_
 httpplib::ClientImpl, 300
ClientImpl
 httpplib::ClientImpl, 207, 208
close
 httpplib::detail::mmap, 351
close_socket
 httpplib::ClientImpl, 210
 httpplib::detail, 35
compareAbs
 BigInt, 145
compress
 httpplib::detail::compressor, 307
 httpplib::detail::nocompressor, 372
compress_
 httpplib::ClientImpl, 301
Compression
 httpplib, 17
cond_
 httpplib::ThreadPool, 527
Conflict_409
 httpplib, 18
Connection
 httpplib, 17
connection_timeout_sec
 httpplib::ClientImpl, 301
connection_timeout_usec
 httpplib::ClientImpl, 301

ConnectionTimeout
 `httpplib`, 17
content
 `httpplib::MultipartFormData`, 357
content_length_
 `httpplib::Request`, 392
 `httpplib::Response`, 409
content_provider_
 `httpplib::detail::ContentProviderAdapter`, 308
 `httpplib::Request`, 392
 `httpplib::Response`, 409
content_provider_resource_releaser_
 `httpplib::Response`, 409
content_provider_success_
 `httpplib::Response`, 409
content_receiver
 `httpplib::Request`, 392
content_type
 `httpplib::MultipartFormData`, 357
 `httpplib::MultipartFormDataProvider`, 370
ContentProvider
 `httpplib`, 14
ContentProviderAdapter
 `httpplib::detail::ContentProviderAdapter`, 308
ContentProviderResourceReleaser
 `httpplib`, 14
ContentProviderWithoutLength
 `httpplib`, 14
ContentReader
 `httpplib::ContentReader`, 310
ContentReceiver
 `httpplib`, 14
ContentReceiverWithProgress
 `httpplib`, 15
Continue_100
 `httpplib`, 18
copy_settings
 `httpplib::ClientImpl`, 211
CPPHTTPPLIB_CLIENT_MAX_TIMEOUT_MSEC
 `httpplib.h`, 545
CPPHTTPPLIB_CLIENT_READ_TIMEOUT_SECONDCPPHTTPPLIB_THREAD_POOL_COUNT
 `httpplib.h`, 545
CPPHTTPPLIB_CLIENT_READ_TIMEOUT_USECOND
 `httpplib.h`, 545
CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_SECOND_and_connect_socket
 `httpplib.h`, 545
CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_USECONDclient_socket
 `httpplib.h`, 545
CPPHTTPPLIB_COMPRESSION_BUFSIZ
 `httpplib.h`, 545
CPPHTTPPLIB_CONNECTION_TIMEOUT_SECONDhttpplib::Server, 443
 `httpplib.h`, 546
CPPHTTPPLIB_CONNECTION_TIMEOUT_USECONDhttpplib::detail, 38
 `httpplib.h`, 546
CPPHTTPPLIB_FORM_URL_ENCODED_PAYLOAD_JWTAX338LENGTH
 `httpplib.h`, 546
CPPHTTPPLIB_HEADER_MAX_LENGTH
 `httpplib.h`, 546
CPPHTTPPLIB_IDLE_INTERVAL_SECOND
 `httpplib.h`, 546
CPPHTTPPLIB_IDLE_INTERVAL_USECOND
 `httpplib.h`, 546
CPPHTTPPLIB_IPV6_V6ONLY
 `httpplib.h`, 546
CPPHTTPPLIB_KEEPALIVE_MAX_COUNT
 `httpplib.h`, 546
CPPHTTPPLIB_KEEPALIVE_TIMEOUT_CHECK_INTERVAL
 `httpplib.h`, 547
CPPHTTPPLIB_KEEPALIVE_TIMEOUT_SECOND
 `httpplib.h`, 547
CPPHTTPPLIB_LISTEN_BACKLOG
 `httpplib.h`, 547
CPPHTTPPLIB_MULTIPART_FORM_DATA_FILE_MAX_CO
 `httpplib.h`, 547
CPPHTTPPLIB_PAYLOAD_MAX_LENGTH
 `httpplib.h`, 547
CPPHTTPPLIB_RANGE_MAX_COUNT
 `httpplib.h`, 547
CPPHTTPPLIB_RECV_BUFSIZ
 `httpplib.h`, 547
CPPHTTPPLIB_RECV_FLAGS
 `httpplib.h`, 547
CPPHTTPPLIB_REDIRECT_MAX_COUNT
 `httpplib.h`, 548
CPPHTTPPLIB_REQUEST_URI_MAX_LENGTH
 `httpplib.h`, 548
CPPHTTPPLIB_SEND_FLAGS
 `httpplib.h`, 548
CPPHTTPPLIB_SERVER_READ_TIMEOUT_SECOND
 `httpplib.h`, 548
CPPHTTPPLIB_SERVER_READ_TIMEOUT_USECOND
 `httpplib.h`, 548
CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_SECOND
 `httpplib.h`, 548
CPPHTTPPLIB_SERVER_WRITE_TIMEOUT_USECOND
 `httpplib.h`, 548
CPPHTTPPLIB_TCP_NODELAY
 `httpplib.h`, 548
CPPHTTPPLIB_THREAD_POOL_COUNT
 `httpplib.h`, 549
CPPHTTPPLIB_VERSION
 `httpplib.h`, 549
CPPHTTPPLIB_CLIENT_WRITE_TIMEOUT_SECONDhttpplib::ClientImpl, 212
 `httpplib.h`, 545
CPPHTTPPLIB_CREATE_SOCKET
 `httpplib::detail`, 36
create_server_socket
 `httpplib::detail`, 36
createAccessAccessToken
 `httpplib::detail`, 38
Created_201
 `httpplib`, 18
createRefreshToken

JWT, 339
crlf_
 httpplib::detail::MultipartFormDataParser, 368
crlf_dash_boundary_
 httpplib::detail::MultipartFormDataParser, 368

d
 RSAPrivateKey, 425
dash_
 httpplib::detail::MultipartFormDataParser, 368
dash_boundary_crlf_
 httpplib::detail::MultipartFormDataParser, 368
data
 httpplib::detail::mmap, 352
data_sink_streambuf
 httpplib::DataSink::data_sink_streambuf, 313
Database, 314
 addUser, 315
 blacklistToken, 315
 cleanupBlacklist, 316
 getUser, 317
 init, 317
 isTokenBlacklisted, 319
Database.cpp, 683
 db, 683
Database.h, 536, 537
DataSink
 httpplib::DataSink, 321, 322
db
 Database.cpp, 683
decode
 Base64URL, 137
decode_url
 httpplib::detail, 40
decommission
 httpplib::Server, 444
decompress
 httpplib::detail::decompressor, 325
decompress_
 httpplib::ClientImpl, 301
decrypt
 RSA, 418
default_file_mimetype_
 httpplib::Server, 489
default_headers_
 httpplib::ClientImpl, 301
 httpplib::Server, 490
default_socket_options
 httpplib, 21
Delete
 httpplib::Client, 173–175
 httpplib::ClientImpl, 215, 217–221
 httpplib::Server, 444
delete_handlers_
 httpplib::Server, 490
delete_handlers_for_content_reader_
 httpplib::Server, 490
digits
 BigInt, 159

dispatch_request
 httpplib::Server, 445
dispatch_request_for_content_reader
 httpplib::Server, 445
divide
 httpplib::detail, 41, 42
done
 httpplib::DataSink, 323
done_with_trailer
 httpplib::DataSink, 323
duration
 httpplib::detail::BufferStream, 163
 httpplib::detail::SocketStream, 503
 httpplib::Stream, 512
duration_to_sec_and_usec
 httpplib::detail, 42

e
 RSAPublicKey, 427
EarlyHints_103
 httpplib, 18
encode
 Base64URL, 138
encode_query_param
 httpplib::detail, 43
encode_url
 httpplib::detail, 44
encoding_type
 httpplib::detail, 45
EncodingType
 httpplib::detail, 32
encrypt
 RSA, 419
end_with_crlf
 httpplib::detail::stream_line_reader, 517
enqueue
 httpplib::TaskQueue, 522
 httpplib::ThreadPool, 526
equal
 httpplib::detail::case_ignore, 125
err_
 httpplib::Result, 417
Error
 httpplib, 16
error
 httpplib::Result, 412
error_handler_
 httpplib::Server, 490
escape_abstract_namespace_unix_domain
 httpplib::detail, 46
ExceedRedirectCount
 httpplib, 17
exception_handler_
 httpplib::Server, 490
ExceptionHandler
 httpplib::Server, 436
execute_on_destruction
 httpplib::detail::scope_exit, 430
exit_function

httpplib::detail::scope_exit, 430
Expect100ContinueHandler
 httpplib::Server, 436
expect_100_continue_handler_
 httpplib::Server, 490
expect_content
 httpplib::detail, 46
ExpectationFailed_417
 httpplib, 18
extractField
 HttpServer.cpp, 685

FailedDependency_424
 httpplib, 19
fd_
 httpplib::detail::mmap, 355
file_
 httpplib::detail::MultipartFormDataParser, 369
file_content_content_type_
 httpplib::Response, 409
file_content_path_
 httpplib::Response, 410
file_extension
 httpplib::detail, 47
file_extension_and_mimetype_map_
 httpplib::Server, 490
file_request_handler_
 httpplib::Server, 490
filename
 httpplib::MultipartFormData, 357
 httpplib::MultipartFormDataProvider, 370
files
 httpplib::Request, 392
FileStat
 httpplib::detail::FileStat, 327
find_content_type
 httpplib::detail, 47
fixed_buffer_
 httpplib::detail::stream_line_reader, 520
fixed_buffer_size_
 httpplib::detail::stream_line_reader, 520
fixed_buffer_used_size_
 httpplib::detail::stream_line_reader, 520
follow_location_
 httpplib::ClientImpl, 301
Forbidden_403
 httpplib, 18
Found_302
 httpplib, 18
from_hex_to_i
 httpplib::detail, 48
from_i_to_hex
 httpplib::detail, 49

GatewayTimeout_504
 httpplib, 19
gcd
 BigInt, 145
generate_keys

RSA, 420
Get
 httpplib::Client, 176–179
 httpplib::ClientImpl, 222–230
 httpplib::Server, 446
get_bearer_token_auth
 httpplib, 21
get_buffer
 httpplib::detail::BufferStream, 163
get_file_value
 httpplib::Request, 382
get_file_values
 httpplib::Request, 383
get_handlers_
 httpplib::Server, 491
get_header_value
 httpplib::detail, 50
 httpplib::Request, 384
 httpplib::Response, 399
get_header_value_count
 httpplib::Request, 384
 httpplib::Response, 400
get_header_value_u64
 httpplib::detail, 51
 httpplib::Request, 385
 httpplib::Response, 400
get_ip_and_port
 httpplib::detail, 52
get_local_ip_and_port
 httpplib::detail, 53
 httpplib::detail::BufferStream, 164
 httpplib::detail::SocketStream, 503
 httpplib::Stream, 512
get_multipart_content_provider
 httpplib::ClientImpl, 230
get_multipart_ranges_data_length
 httpplib::detail, 54
get_param_value
 httpplib::Request, 386
get_param_value_count
 httpplib::Request, 387
get_range_offset_and_length
 httpplib::detail, 55
get_remote_ip_and_port
 httpplib::detail, 55
 httpplib::detail::BufferStream, 164
 httpplib::detail::SocketStream, 503
 httpplib::Stream, 512
get_request_header_value
 httpplib::Result, 412
get_request_header_value_count
 httpplib::Result, 413
get_request_header_value_u64
 httpplib::Result, 414
getline
 httpplib::detail::stream_line_reader, 518
getUser
 Database, 317

Gone_410
 `httpplib`, 18
growable_buffer_
 `httpplib::detail::stream_line_reader`, 520
Gzip
 `httpplib::detail`, 32

handle_EINTR
 `httpplib::detail`, 56
handle_file_request
 `httpplib::Server`, 447
handle_request
 `httpplib::ClientImpl`, 232
Handled
 `httpplib::Server`, 437
Handler
 `httpplib::Server`, 436
HandlerResponse
 `httpplib::Server`, 437
Handlers
 `httpplib::Server`, 436
HandlersForContentReader
 `httpplib::Server`, 436
HandlerWithContentReader
 `httpplib::Server`, 436
HandlerWithResponse
 `httpplib::Server`, 437
has_crlf
 `httpplib::detail`, 57
has_file
 `httpplib::Request`, 387
has_header
 `httpplib::detail`, 58
 `httpplib::Request`, 388
 `httpplib::Response`, 401
has_param
 `httpplib::Request`, 389
has_request_header
 `httpplib::Result`, 414
hash
 SHA256, 495
hash_core
 `httpplib::detail::case_ignore::hash`, 329
hashPassword
 `PasswordEncryptor`, 373
Head
 `httpplib::Client`, 179
 `httpplib::ClientImpl`, 234, 235
header_writer_
 `httpplib::ClientImpl`, 301
 `httpplib::Server`, 491
Headers
 `httpplib`, 15
headers
 `httpplib::Request`, 392
 `httpplib::Response`, 410
 `httpplib::Server::MountPointEntry`, 356
host
 `httpplib::Client`, 179

 `httpplib::ClientImpl`, 236
host_
 `httpplib::ClientImpl`, 302
host_and_port_
 `httpplib::ClientImpl`, 302
hosted_at
 `httpplib`, 22
httpplib, 12
 Accepted_202, 18
 AlreadyReported_208, 18
 append_query_params, 20
 BadGateway_502, 19
 BadRequest_400, 18
 BindIPAddress, 17
 Canceled, 17
 CertificateAccepted, 17
 CertificateRejected, 17
 Compression, 17
 Conflict_409, 18
 Connection, 17
 ConnectionTimeout, 17
 ContentProvider, 14
 ContentProviderResourceReleaser, 14
 ContentProviderWithoutLength, 14
 ContentReceiver, 14
 ContentReceiverWithProgress, 15
 Continue_100, 18
 Created_201, 18
 default_socket_options, 21
 EarlyHints_103, 18
 Error, 16
 ExceedRedirectCount, 17
 ExpectationFailed_417, 18
 FailedDependency_424, 19
 Forbidden_403, 18
 Found_302, 18
 GatewayTimeout_504, 19
 get_bearer_token_auth, 21
 Gone_410, 18
 Headers, 15
 hosted_at, 22
 HttpVersionNotSupported_505, 19
 ImATeapot_418, 19
 IMUsed_226, 18
 InsufficientStorage_507, 19
 InternalServerError_500, 19
 LengthRequired_411, 18
 Locked_423, 19
 Logger, 15
 LoopDetected_508, 19
 make_basic_authentication_header, 23
 make_bearer_token_authentication_header,
 24
 make_range_header, 25
 Match, 15
 MethodNotAllowed_405, 18
 MisdirectedRequest_421, 19
 MovedPermanently_301, 18

- MultipartContentHeader, 15
MultipartFormDataItems, 15
MultipartFormDataMap, 16
MultipartFormDataProviderItems, 16
MultipleChoices_300, 18
MultiStatus_207, 18
NetworkAuthenticationRequired_511, 19
NoContent_204, 18
NoDecisionMade, 17
NonAuthoritativeInformation_203, 18
NotAcceptable_406, 18
NotExtended_510, 19
NotFound_404, 18
NotImplemented_501, 19
NotModified_304, 18
OK_200, 18
operator<<, 25
Params, 16
PartialContent_206, 18
PayloadTooLarge_413, 18
PaymentRequired_402, 18
PermanentRedirect_308, 18
PreconditionFailed_412, 18
PreconditionRequired_428, 19
Processing_102, 18
Progress, 16
ProxyAuthenticationRequired_407, 18
ProxyConnection, 17
Range, 16
RangeNotSatisfiable_416, 18
Ranges, 16
Read, 17
RequestHeaderFieldsTooLarge_431, 19
RequestTimeout_408, 18
ResetContent_205, 18
ResponseHandler, 16
SeeOther_303, 18
ServiceUnavailable_503, 19
SocketOptions, 16
SSLConnection, 17
SSLLoadingCerts, 17
SSLPeerCouldBeClosed_, 17
SSLServerHostnameVerification, 17
SSLServerVerification, 17
SSLVerifierResponse, 17
status_message, 26
StatusCode, 17
Success, 17
SwitchingProtocol_101, 18
TemporaryRedirect_307, 18
to_string, 27
TooEarly_425, 19
TooManyRequests_429, 19
Unauthorized_401, 18
UnavailableForLegalReasons_451, 19
Unknown, 17
UnprocessableContent_422, 19
UnsupportedMediaType_415, 18
- UnsupportedMultipartBoundaryChars, 17
unused_306, 18
UpgradeRequired_426, 19
UriTooLong_414, 18
UseProxy_305, 18
VariantAlsoNegotiates_506, 19
Write, 17
httpplib.h, 537, 550
CPPHTTPLIB_CLIENT_MAX_TIMEOUT_MSECOND,
545
CPPHTTPLIB_CLIENT_READ_TIMEOUT_SECOND,
545
CPPHTTPLIB_CLIENT_READ_TIMEOUT_USECOND,
545
CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_SECOND,
545
CPPHTTPLIB_CLIENT_WRITE_TIMEOUT_USECOND,
545
CPPHTTPLIB_COMPRESSION_BUFSIZ,
545
CPPHTTPLIB_CONNECTION_TIMEOUT_SECOND,
546
CPPHTTPLIB_CONNECTION_TIMEOUT_USECOND,
546
CPPHTTPLIB_FORM_URL_ENCODED_PAYLOAD_MA
546
CPPHTTPLIB_HEADER_MAX_LENGTH,
546
CPPHTTPLIB_IDLE_INTERVAL_SECOND,
546
CPPHTTPLIB_IDLE_INTERVAL_USECOND,
546
CPPHTTPLIB_IPV6_V6ONLY, 546
CPPHTTPLIB_KEEPALIVE_MAX_COUNT,
546
CPPHTTPLIB_KEEPALIVE_TIMEOUT_CHECK_INTER
547
CPPHTTPLIB_KEEPALIVE_TIMEOUT_SECOND,
547
CPPHTTPLIB_LISTEN_BACKLOG, 547
CPPHTTPLIB_MULTIPART_FORM_DATA_FILE_MAX
547
CPPHTTPLIB_PAYLOAD_MAX_LENGTH,
547
CPPHTTPLIB_RANGE_MAX_COUNT,
547
CPPHTTPLIB_RECV_BUFSIZ, 547
CPPHTTPLIB_RECV_FLAGS, 547
CPPHTTPLIB_REDIRECT_MAX_COUNT,
548
CPPHTTPLIB_REQUEST_URI_MAX_LENGTH,
548
CPPHTTPLIB_SEND_FLAGS, 548
CPPHTTPLIB_SERVER_READ_TIMEOUT_SECOND,
548
CPPHTTPLIB_SERVER_READ_TIMEOUT_USECOND,
548
CPPHTTPLIB_SERVER_WRITE_TIMEOUT_SECOND,

548
CPPHTTPLIB_SERVER_WRITE_TIMEOUT_USEC, 300
548
CPPHTTPLIB_TCP_NODELAY, 548
CPPHTTPLIB_THREAD_POOL_COUNT, 549
549
CPPHTTPLIB_VERSION, 549
INVALID_SOCKET, 549
socket_t, 549
USE_IF2IP, 549
http://Client, 166
~Client, 173
cli_, 200
Client, 170–173
Delete, 173–175
Get, 176–179
Head, 179
host, 179
is_socket_open, 180
is_valid, 180
operator=, 180
Options, 181
Patch, 181–184
port, 184
Post, 185–189
Put, 190–194
send, 194
set_address_family, 194
set_basic_auth, 195
set_bearer_token_auth, 195
set_compress, 195
set_connection_timeout, 195
set_decompress, 196
set_default_headers, 196
set_follow_location, 196
set_header_writer, 196
set_hostname_addr_map, 196
set_interface, 197
set_keep_alive, 197
set_logger, 197
set_max_timeout, 197
set_proxy, 198
set_proxy_basic_auth, 198
set_proxy_bearer_token_auth, 198
set_read_timeout, 199
set_socket_options, 199
set_tcp_nodelay, 199
set_url_encode, 199
set_write_timeout, 200
socket, 200
stop, 200
http://ClientImpl, 201
~ClientImpl, 208
addr_map_, 300
address_family_, 300
adjust_host_string, 209
basic_auth_password_, 300
basic_auth_username_, 300
bearer_token_auth_token_, 300
CERT_PATH, 300
client_key_path_, 300
ClientImpl, 207, 208
close_socket, 210
compress_, 301
connection_timeout_sec_, 301
connection_timeout_usec_, 301
copy_settings, 211
create_and_connect_socket, 212
create_client_socket, 214
decompress_, 301
default_headers_, 301
Delete, 215, 217–221
follow_location_, 301
Get, 222–230
get_multipart_content_provider, 230
handle_request, 232
Head, 234, 235
header_writer_, 301
host, 236
host_, 302
host_and_port_, 302
interface_, 302
ipv6_v6only_, 302
is_socket_open, 236
is_ssl, 236
is_valid, 237
keep_alive_, 302
logger_, 302
max_timeout_msec_, 302
Options, 237, 238
Patch, 239–246
port, 246
port_, 303
Post, 247–258
process_request, 258
process_socket, 261
proxy_basic_auth_password_, 303
proxy_basic_auth_username_, 303
proxy_bearer_token_auth_token_, 303
proxy_host_, 303
proxy_port_, 303
Put, 262–273
read_response_line, 273
read_timeout_sec_, 303
read_timeout_usec_, 303
redirect, 274
request_mutex_, 304
send, 276, 277
send_, 278, 279
send_with_content_provider, 281, 282
set_address_family, 284
set_basic_auth, 284
set_bearer_token_auth, 285
set_compress, 285
set_connection_timeout, 285
set_decompress, 286

set_default_headers, 286
set_follow_location, 286
set_header_writer, 286
set_hostname_addr_map, 287
set_interface, 287
set_ipv6_v6only, 287
set_keep_alive, 287
set_logger, 287
set_max_timeout, 287, 288
set_proxy, 288
set_proxy_basic_auth, 289
set_proxy_bearer_token_auth, 289
set_read_timeout, 289, 290
set_socket_options, 290
set_tcp_nodelay, 291
set_url_encode, 291
set_write_timeout, 291
shutdown_socket, 292
shutdown_ssl, 293
socket, 294
socket_, 304
socket_mutex_, 304
socket_options_, 304
socket_requests_are_from_thread_, 304
socket_requests_in_flight_, 304
socket_should_be_closed_when_request_is_done_ 304
stop, 295
tcp_nodelay_, 304
url_encode_, 305
write_content_with_provider, 296
write_request, 297
write_timeout_sec_, 305
write_timeout_usec_, 305
httpplib::ClientImpl::Socket, 497
 is_open, 498
 sock, 498
httpplib::ContentReader, 309
 ContentReader, 310
 multipart_reader_, 310
 MultipartReader, 309
 operator(), 310
 Reader, 309
 reader_, 310
httpplib::DataSink, 320
 DataSink, 321, 322
 done, 323
 done_with_trailer, 323
 is_writable, 323
 operator=, 322
 os, 323
 sb_, 323
 write, 323
httpplib::DataSink::data_sink_streambuf, 311
 data_sink_streambuf, 313
 sink_, 313
 xputn, 313
httpplib::detail, 28
base64_encode, 32
bind_ip_address, 32
Brotli, 32
calc_actual_timeout, 33
can_compress_content_type, 34
close_socket, 35
create_client_socket, 36
create_socket, 38
decode_url, 40
divide, 41, 42
duration_to_sec_and_usec, 42
encode_query_param, 43
encode_url, 44
encoding_type, 45
EncodingType, 32
escape_abstract_namespace_unix_domain, 46
expect_content, 46
file_extension, 47
find_content_type, 47
from_hex_to_i, 48
from_i_to_hex, 49
get_header_value, 50
get_header_value_u64, 51
get_ip_and_port, 52
get_local_ip_and_port, 53
get_multipart_ranges_data_length, 54
get_range_offset_and_length, 55
get_remote_ip_and_port, 55
Gzip, 32
handle_EINTR, 56
has_crlf, 57
has_header, 58
if2ip, 59
is_chunked_transfer_encoding, 60
is_connection_error, 61
is_hex, 61
is_multipart_boundary_chars_valid, 62
is_numeric, 63
is_socket_alive, 63
is_space_or_tab, 64
is_valid_path, 65
keep_alive, 65
make_content_range_header_field, 66
make_multipart_data_boundary, 67
make_multipart_ranges_data, 68
make_unique, 68, 69
None, 32
params_to_query_str, 69
parse_disposition_params, 70
parse_header, 71
parse_multipart_boundary, 72
parse_query_text, 73, 74
parse_range_header, 74
parse_www_authenticate, 76
poll_wrapper, 77
prepare_content_receiver, 77
process_client_socket, 79

process_multipart_ranges_data, 80
process_server_socket, 81
process_server_socket_core, 82
random_string, 83
range_error, 84
read_content, 85
read_content_chunked, 86
read_content_with_length, 88
read_content_without_length, 89
read_headers, 90
read_socket, 91
redirect, 92
select_impl, 93
select_read, 94
select_write, 95
send_socket, 96
serialize_multipart_formdata, 97
serialize_multipart_formdata_finish, 98
serialize_multipart_formdata_get_content_type, 99
serialize_multipart_formdata_item_begin, 100
serialize_multipart_formdata_item_end, 101
set_nonblocking, 102
set_socket_opt, 103
set_socket_opt_impl, 103
set_socket_opt_time, 104
shutdown_socket, 105
skip_content_with_length, 106
split, 107, 108
str2tag, 108
str2tag_core, 109
str_len, 110
to_utf8, 110
trim, 111
trim_copy, 112
trim_double_quotes_copy, 112
unescape_abstract_namespace_unix_domain, 113
wait_until_socket_is_ready, 113
write_content, 115
write_content_chunked, 117
write_content_without_length, 119
write_data, 120
write_headers, 121
write_multipart_ranges_data, 122
write_request_line, 123
write_response_line, 124
Zstd, 32

httpplib::detail::BufferStream, 160
~BufferStream, 163
buffer, 166
BufferStream, 163
duration, 163
get_buffer, 163
get_local_ip_and_port, 164
get_remote_ip_and_port, 164
is_readable, 164

position, 166
read, 164
socket, 165
wait_readable, 165
wait_writable, 165
write, 165

httpplib::detail::case_ignore, 125
equal, 125
to_lower, 125

httpplib::detail::case_ignore::equal_to, 326
operator(), 326

httpplib::detail::case_ignore::hash, 329
hash_core, 329
operator(), 330

httpplib::detail::compressor, 305
~compressor, 306
Callback, 306
compress, 307

httpplib::detail::ContentProviderAdapter, 307
content_provider_, 308
ContentProviderAdapter, 308
operator(), 308

httpplib::detail::decompressor, 324
~decompressor, 325
Callback, 325
decompress, 325
is_valid, 325

httpplib::detail::fields, 126
is_field_content, 126
is_field_name, 127
is_field_value, 128
is_field_vchar, 130
is_obs_text, 131
is_token, 131
is_token_char, 132
is_vchar, 133

httpplib::detail::FileStat, 327
FileStat, 327
is_dir, 328
is_file, 328
ret_, 328
st_, 328

httpplib::detail::MatcherBase, 348
~MatcherBase, 349
match, 349

httpplib::detail::mmap, 349
~mmap, 351
addr_, 355
close, 351
data, 352
fd_, 355
is_open, 352
is_open_empty_file, 355
mmap, 350
open, 352
size, 354
size_, 355

httpplib::detail::MultipartFormDataParser, 358

boundary_, 368
buf_, 368
buf_append, 360
buf_data, 360
buf_epos_, 368
buf_erase, 360
buf_find, 361
buf_head, 362
buf_size, 362
buf_spos_, 368
buf_start_with, 362
clear_file_info, 363
crlf_, 368
crlf_dash_boundary_, 368
dash_, 368
dash_boundary_crlf_, 368
file_, 369
is_valid, 363
is_valid_, 369
MultipartFormDataParser, 359
parse, 363
set_boundary, 366
start_with, 366
start_with_case_ignore, 367
state_, 369
httplib::detail::nocompressor, 371
~nocompressor, 372
compress, 372
httplib::detail::PathParamsMatcher, 374
match, 377
param_names_, 378
PathParamsMatcher, 376
separator, 378
static_fragments_, 378
httplib::detail::RegexMatcher, 379
match, 380
regex_, 380
RegexMatcher, 380
httplib::detail::scope_exit, 427
~scope_exit, 429
execute_on_destruction, 430
exit_function, 430
operator=, 430
release, 430
scope_exit, 428, 429
httplib::detail::SocketStream, 498
~SocketStream, 502
duration, 503
get_local_ip_and_port, 503
get_remote_ip_and_port, 503
is_readable, 504
max_timeout_msec_, 507
read, 504
read_buff_, 507
read_buff_content_size_, 507
read_buff_off_, 508
read_buff_size_, 508
read_timeout_sec_, 508
read_timeout_usec_, 508
sock_, 508
socket, 505
SocketStream, 502
start_time_, 508
wait_readable, 505
wait_writable, 506
write, 507
write_timeout_sec_, 508
write_timeout_usec_, 508
httpplib::detail::stream_line_reader, 515
append, 517
end_with_crlf, 517
fixed_buffer_, 520
fixed_buffer_size_, 520
fixed_buffer_used_size_, 520
getline, 518
growable_buffer_, 520
ptr, 519
size, 519
stream_line_reader, 517
strm_, 520
httpplib::detail::udl, 134
httpplib::MultipartFormData, 356
content, 357
content_type, 357
filename, 357
name, 357
httpplib::MultipartFormDataProvider, 369
content_type, 370
filename, 370
name, 370
provider, 370
httplib::Request, 381
authorization_count_, 391
body, 391
content_length_, 392
content_provider_, 392
content_receiver, 392
files, 392
get_file_value, 382
get_file_values, 383
get_header_value, 384
get_header_value_count, 384
get_header_value_u64, 385
get_param_value, 386
get_param_value_count, 387
has_file, 387
has_header, 388
has_param, 389
headers, 392
is_chunked_content_provider_, 392
is_connection_closed, 392
is_multipart_form_data, 389
local_addr, 392
local_port, 393
matches, 393
method, 393

params, 393
path, 393
path_params, 393
progress, 393
ranges, 393
redirect_count_, 394
remote_addr, 394
remote_port, 394
response_handler, 394
set_header, 390
start_time_, 394
target, 394
version, 394
httplib::Response, 395
 ~Response, 399
 body, 409
 content_length_, 409
 content_provider_, 409
 content_provider_resource_releaser_, 409
 content_provider_success_, 409
 file_content_content_type_, 409
 file_content_path_, 410
 get_header_value, 399
 get_header_value_count, 400
 get_header_value_u64, 400
 has_header, 401
 headers, 410
 is_chunked_content_provider_, 410
 location, 410
 operator=, 402
 reason, 410
 Response, 397, 398
 set_chunked_content_provider, 402
 set_content, 403, 404
 set_content_provider, 405
 set_file_content, 406, 407
 set_header, 407
 set_redirect, 408
 status, 410
 version, 410
httplib::Result, 411
 err_, 417
 error, 412
 get_request_header_value, 412
 get_request_header_value_count, 413
 get_request_header_value_u64, 414
 has_request_header, 414
 operator bool, 415
 operator!=, 415
 operator->, 416
 operator==, 416
 operator*, 415, 416
 request_headers_, 417
 res_, 417
 Result, 412
 value, 416
httplib::Server, 431
 ~Server, 438
 address_family_, 489
 apply_ranges, 439
 base_dirs_, 489
 bind_internal, 441
 bind_to_any_port, 442
 bind_to_port, 442
 create_server_socket, 443
 decommission, 444
 default_file_mimetype_, 489
 default_headers_, 490
 Delete, 444
 delete_handlers_, 490
 delete_handlers_for_content_reader_, 490
 dispatch_request, 445
 dispatch_request_for_content_reader, 445
 error_handler_, 490
 exception_handler_, 490
 ExceptionHandler, 436
 Expect100ContinueHandler, 436
 expect_100_continue_handler_, 490
 file_extension_and_mimetype_map_, 490
 file_request_handler_, 490
 Get, 446
 get_handlers_, 491
 handle_file_request, 447
 Handled, 437
 Handler, 436
 HandlerResponse, 437
 Handlers, 436
 HandlersForContentReader, 436
 HandlerWithContentReader, 436
 HandlerWithResponse, 437
 header_writer_, 491
 idle_interval_sec_, 491
 idle_interval_usec_, 491
 ipv6_v6only_, 491
 is_decommissioned, 491
 is_running, 448
 is_running_, 491
 is_valid, 448
 keep_alive_max_count_, 492
 keep_alive_timeout_sec_, 492
 listen, 448
 listen_after_bind, 449
 listen_internal, 449
 logger_, 492
 make_matcher, 451
 new_task_queue, 492
 Options, 452
 options_handlers_, 492
 parse_request_line, 453
 Patch, 454
 patch_handlers_, 492
 patch_handlers_for_content_reader_, 492
 payload_max_length_, 492
 Post, 455, 456
 post_handlers_, 493
 post_handlers_for_content_reader_, 493

post_routing_handler_, 493
pre_routing_handler_, 493
process_and_close_socket, 456
process_request, 457
Put, 461
put_handlers_, 493
put_handlers_for_content_reader_, 493
read_content, 462
read_content_core, 463
read_content_with_content_receiver, 465
read_timeout_sec_, 493
read_timeout_usec_, 493
remove_mount_point, 466
routing, 466
Server, 437
set_address_family, 468
set_base_dir, 469
set_default_file_mimetype, 469
set_default_headers, 470
set_error_handler, 470
set_error_handler_core, 471
set_exception_handler, 472
set_expect_100_continue_handler, 472
set_file_extension_and_mimetype_mapping, 473
set_file_request_handler, 473
set_header_writer, 473
set_idle_interval, 474
set_ipv6_v6only, 475
set_keep_alive_max_count, 476
set_keep_alive_timeout, 476
set_logger, 477
set_mount_point, 477
set_payload_max_length, 478
set_post_routing_handler, 478
set_pre_routing_handler, 479
set_read_timeout, 479, 480
set_socket_options, 481
set_tcp_nodelay, 481
set_write_timeout, 482
socket_options_, 494
stop, 483
svr_sock_, 494
tcp_nodelay_, 494
Unhandled, 437
wait_until_ready, 483
write_content_with_provider, 483
write_response, 485
write_response_core, 486
write_response_with_content, 488
write_timeout_sec_, 494
write_timeout_usec_, 494
httplib::Server::MountPointEntry, 355
base_dir, 356
headers, 356
mount_point, 356
httplib::Stream, 509
~Stream, 511
duration, 512
get_local_ip_and_port, 512
get_remote_ip_and_port, 512
is_readable, 512
read, 512
socket, 512
wait_readable, 513
wait_writable, 513
write, 513, 514
httplib::TaskQueue, 521
~TaskQueue, 522
enqueue, 522
on_idle, 522
shutdown, 523
TaskQueue, 522
httplib::ThreadPool, 523
~ThreadPool, 526
cond_, 527
enqueue, 526
jobs_, 527
max_queued_requests_, 527
mutex_, 527
shutdown, 526
shutdown_, 527
ThreadPool, 525, 526
threads_, 528
worker, 527
httplib::ThreadPool::worker, 530
operator(), 531
pool_, 532
worker, 531
HttpServer, 331
start, 331
HttpServer.cpp, 685, 686
extractField, 685
HttpServer.h, 670, 671
HttpVersionNotSupported_505
 httplib, 19
id
 User, 529
idle_interval_sec_
 httplib::Server, 491
idle_interval_usec_
 httplib::Server, 491
if2ip
 httplib::detail, 59
ImATeaPot_418
 httplib, 19
IMUsed_226
 httplib, 18
init
 Database, 317
InsufficientStorage_507
 httplib, 19
interface_
 httplib::ClientImpl, 302
InternalServerError_500
 httplib, 19

INVALID_SOCKET
 `httpplib.h`, 549

ipv6_v6only_
 `httpplib::ClientImpl`, 302
 `httpplib::Server`, 491

is_chunked_content_provider_
 `httpplib::Request`, 392
 `httpplib::Response`, 410

is_chunked_transfer_encoding
 `httpplib::detail`, 60

is_connection_closed
 `httpplib::Request`, 392

is_connection_error
 `httpplib::detail`, 61

is_decommissioned
 `httpplib::Server`, 491

is_dir
 `httpplib::detail::FileStat`, 328

is_field_content
 `httpplib::detail::fields`, 126

is_field_name
 `httpplib::detail::fields`, 127

is_field_value
 `httpplib::detail::fields`, 128

is_field_vchar
 `httpplib::detail::fields`, 130

is_file
 `httpplib::detail::FileStat`, 328

is_hex
 `httpplib::detail`, 61

is_multipart_boundary_chars_valid
 `httpplib::detail`, 62

is_multipart_form_data
 `httpplib::Request`, 389

is_numeric
 `httpplib::detail`, 63

is_obs_text
 `httpplib::detail::fields`, 131

is_open
 `httpplib::ClientImpl::Socket`, 498
 `httpplib::detail::mmap`, 352

is_open_empty_file
 `httpplib::detail::mmap`, 355

is_prime
 `RSA.cpp`, 698

is_readable
 `httpplib::detail::BufferStream`, 164
 `httpplib::detail::SocketStream`, 504
 `httpplib::Stream`, 512

is_running
 `httpplib::Server`, 448

is_running_
 `httpplib::Server`, 491

is_socket_alive
 `httpplib::detail`, 63

is_socket_open
 `httpplib::Client`, 180
 `httpplib::ClientImpl`, 236

is_space_or_tab
 `httpplib::detail`, 64

is_ssl
 `httpplib::ClientImpl`, 236

is_token
 `httpplib::detail::fields`, 131

is_token_char
 `httpplib::detail::fields`, 132

is_valid
 `httpplib::Client`, 180
 `httpplib::ClientImpl`, 237
 `httpplib::detail::decompressor`, 325
 `httpplib::detail::MultipartFormDataParser`, 363
 `httpplib::Server`, 448

is_valid_
 `httpplib::detail::MultipartFormDataParser`, 369

is_valid_path
 `httpplib::detail`, 65

is_vchar
 `httpplib::detail::fields`, 133

is_writable
 `httpplib::DataSink`, 323

isNegative
 `BigInt`, 146

isTokenBlacklisted
 `Database`, 319

isZero
 `BigInt`, 147

jobs_
 `httpplib::ThreadPool`, 527

JWT, 336
 `createAccessToken`, 338
 `createRefreshToken`, 339
 `verifyAccessToken`, 340
 `verifyRefreshToken`, 342

JWT.cpp, 690

JWT.h, 671, 672

k
 `anonymous_namespace{SHA256.cpp}`, 12

keep_alive
 `httpplib::detail`, 65

keep_alive_
 `httpplib::ClientImpl`, 302

keep_alive_max_count_
 `httpplib::Server`, 492

keep_alive_timeout_sec_
 `httpplib::Server`, 492

KeyStorage, 344
 `loadKeys`, 345
 `saveKeys`, 346

KeyStorage.cpp, 693, 694
 `PRIV_FILE`, 694
 `PUB_FILE`, 694

KeyStorage.h, 672, 673

LengthRequired_411
 `httpplib`, 18

listen
 `httpplib::Server`, 448
listen_after_bind
 `httpplib::Server`, 449
listen_internal
 `httpplib::Server`, 449
loadKeys
 `KeyStorage`, 345
local_addr
 `httpplib::Request`, 392
local_port
 `httpplib::Request`, 393
location
 `httpplib::Response`, 410
Locked_423
 `httpplib`, 19
Logger
 `httpplib`, 15
logger_
 `httpplib::ClientImpl`, 302
 `httpplib::Server`, 492
LoopDetected_508
 `httpplib`, 19

main
 `main.cpp`, 695
main.cpp, 695, 696
 `main`, 695
maj
 `anonymous_namespace{SHA256.cpp}`, 9
make_basic_authentication_header
 `httpplib`, 23
make_bearer_token_authentication_header
 `httpplib`, 24
make_content_range_header_field
 `httpplib::detail`, 66
make_matcher
 `httpplib::Server`, 451
make_multipart_data_boundary
 `httpplib::detail`, 67
make_multipart_ranges_data
 `httpplib::detail`, 68
make_range_header
 `httpplib`, 25
make_unique
 `httpplib::detail`, 68, 69
Match
 `httpplib`, 15
match
 `httpplib::detail::MatcherBase`, 349
 `httpplib::detail::PathParamsMatcher`, 377
 `httpplib::detail::RegexMatcher`, 380
matches
 `httpplib::Request`, 393
max_queued_requests_
 `httpplib::ThreadPool`, 527
max_timeout_msec_
 `httpplib::ClientImpl`, 302
 `httpplib::detail::SocketStream`, 507
method
 `httpplib::Request`, 393
MethodNotAllowed_405
 `httpplib`, 18
MisdirectedRequest_421
 `httpplib`, 19
mmap
 `httpplib::detail::mmap`, 350
modinv
 `RSA.cpp`, 699
modPow
 `BigInt`, 147
mount_point
 `httpplib::Server::MountPointEntry`, 356
MovedPermanently_301
 `httpplib`, 18
multipart_reader_
 `httpplib::ContentReader`, 310
MultipartContentHeader
 `httpplib`, 15
MultipartFormDataItems
 `httpplib`, 15
MultipartFormDataMap
 `httpplib`, 16
MultipartFormDataParser
 `httpplib::detail::MultipartFormDataParser`, 359
MultipartFormDataProviderItems
 `httpplib`, 16
MultipartReader
 `httpplib::ContentReader`, 309
MultipleChoices_300
 `httpplib`, 18
MultiStatus_207
 `httpplib`, 18
mutex_
 `httpplib::ThreadPool`, 527

n
 `RSAPrivatekey`, 425
 `RSApublickey`, 427
name
 `httpplib::MultipartFormData`, 357
 `httpplib::MultipartFormDataProvider`, 370
negative
 `BigInt`, 159
NetworkAuthenticationRequired_511
 `httpplib`, 19
new_task_queue
 `httpplib::Server`, 492
NoContent_204
 `httpplib`, 18
NoDecisionMade
 `httpplib`, 17
NonAuthoritativeInformation_203
 `httpplib`, 18
None
 `httpplib::detail`, 32
NotAcceptable_406
 `httpplib`, 18

NotExtended_ 510
 httpplib, 19
NotFound_ 404
 httpplib, 18
NotImplemented_ 501
 httpplib, 19
NotModified_ 304
 httpplib, 18

OK_ 200
 httpplib, 18
on _idle
 httpplib::TaskQueue, 522
open
 httpplib::detail::mmap, 352
operator bool
 httpplib::Result, 415
operator!=
 BigInt, 149
 httpplib::Result, 415
operator<
 BigInt, 154
operator<<
 httpplib, 25
operator<=
 BigInt, 155
operator>
 BigInt, 156
operator>=
 BigInt, 156
operator()
 httpplib::ContentReader, 310
 httpplib::detail::case_ignore::equal_to, 326
 httpplib::detail::case_ignore::hash, 330
 httpplib::detail::ContentProviderAdapter, 308
 httpplib::ThreadPool::worker, 531
operator+
 BigInt, 151
operator-
 BigInt, 152
operator->
 httpplib::Result, 416
operator/
 BigInt, 153
operator=
 httpplib::Client, 180
 httpplib::DataSink, 322
 httpplib::detail::scope_exit, 430
 httpplib::Response, 402
operator==
 BigInt, 155
 httpplib::Result, 416
operator%
 BigInt, 149
operator*
 BigInt, 150
 httpplib::Result, 415, 416
Options
 httpplib::Client, 181

httpplib::ClientImpl, 237, 238
httpplib::Server, 452
options_handlers_
 httpplib::Server, 492
os
 httpplib::DataSink, 323

pad
 anonymous_namespace{SHA256.cpp}, 9
param_names_
 httpplib::detail::PathParamsMatcher, 378
Params
 httpplib, 16
params
 httpplib::Request, 393
params_to_query_str
 httpplib::detail, 69
parse
 httpplib::detail::MultipartFormDataParser, 363
parse_disposition_params
 httpplib::detail, 70
parse_header
 httpplib::detail, 71
parse_multipart_boundary
 httpplib::detail, 72
parse_query_text
 httpplib::detail, 73, 74
parse_range_header
 httpplib::detail, 74
parse_request_line
 httpplib::Server, 453
parse_www_authenticate
 httpplib::detail, 76
PartialContent_ 206
 httpplib, 18
password
 User, 529
PasswordEncryptor, 373
 hashPassword, 373
PasswordEncryptor.cpp, 696, 697
PasswordEncryptor.h, 673, 674
Patch
 httpplib::Client, 181–184
 httpplib::ClientImpl, 239–246
 httpplib::Server, 454
patch_handlers_
 httpplib::Server, 492
patch_handlers_for_content_reader_
 httpplib::Server, 492
path
 httpplib::Request, 393
path_params
 httpplib::Request, 393
PathParamsMatcher
 httpplib::detail::PathParamsMatcher, 376
payload_max_length_
 httpplib::Server, 492
PayloadTooLarge_ 413
 httpplib, 18

PaymentRequired_402
 `httpplib`, 18
PermanentRedirect_308
 `httpplib`, 18
poll_wrapper
 `httpplib::detail`, 77
pool_
 `httpplib::ThreadPool::worker`, 532
port
 `httpplib::Client`, 184
 `httpplib::ClientImpl`, 246
port_
 `httpplib::ClientImpl`, 303
position
 `httpplib::detail::BufferStream`, 166
Post
 `httpplib::Client`, 185–189
 `httpplib::ClientImpl`, 247–258
 `httpplib::Server`, 455, 456
post_handlers_
 `httpplib::Server`, 493
post_handlers_for_content_reader_
 `httpplib::Server`, 493
post_routing_handler_
 `httpplib::Server`, 493
pre_routing_handler_
 `httpplib::Server`, 493
PreconditionFailed_412
 `httpplib`, 18
PreconditionRequired_428
 `httpplib`, 19
prepare_content_receiver
 `httpplib::detail`, 77
PRIV_FILE
 `KeyStorage.cpp`, 694
process_and_close_socket
 `httpplib::Server`, 456
process_client_socket
 `httpplib::detail`, 79
process_multipart_ranges_data
 `httpplib::detail`, 80
process_request
 `httpplib::ClientImpl`, 258
 `httpplib::Server`, 457
process_server_socket
 `httpplib::detail`, 81
process_server_socket_core
 `httpplib::detail`, 82
process_socket
 `httpplib::ClientImpl`, 261
Processing_102
 `httpplib`, 18
Progress
 `httpplib`, 16
progress
 `httpplib::Request`, 393
provider
 `httpplib::MultipartFormDataProvider`, 370
proxy_basic_auth_password_
 `httpplib::ClientImpl`, 303
proxy_basic_auth_username_
 `httpplib::ClientImpl`, 303
proxy_bearer_token_auth_token_
 `httpplib::ClientImpl`, 303
proxy_host_
 `httpplib::ClientImpl`, 303
proxy_port_
 `httpplib::ClientImpl`, 303
ProxyAuthenticationRequired_407
 `httpplib`, 18
ProxyConnection
 `httpplib`, 17
ptr
 `httpplib::detail::stream_line_reader`, 519
PUB_FILE
 `KeyStorage.cpp`, 694
Put
 `httpplib::Client`, 190–194
 `httpplib::ClientImpl`, 262–273
 `httpplib::Server`, 461
put_handlers_
 `httpplib::Server`, 493
put_handlers_for_content_reader_
 `httpplib::Server`, 493
random_bigint
 `RSA.cpp`, 700
random_string
 `httpplib::detail`, 83
Range
 `httpplib`, 16
range_error
 `httpplib::detail`, 84
RangeNotSatisfiable_416
 `httpplib`, 18
Ranges
 `httpplib`, 16
ranges
 `httpplib::Request`, 393
Read
 `httpplib`, 17
read
 `httpplib::detail::BufferStream`, 164
 `httpplib::detail::SocketStream`, 504
 `httpplib::Stream`, 512
read_buff_
 `httpplib::detail::SocketStream`, 507
read_buff_content_size_
 `httpplib::detail::SocketStream`, 507
read_buff_off_
 `httpplib::detail::SocketStream`, 508
read_buff_size_
 `httpplib::detail::SocketStream`, 508
read_content
 `httpplib::detail`, 85
 `httpplib::Server`, 462
read_content_chunked

httpplib::detail, 86
read_content_core
 httpplib::Server, 463
read_content_with_content_receiver
 httpplib::Server, 465
read_content_with_length
 httpplib::detail, 88
read_content_without_length
 httpplib::detail, 89
read_headers
 httpplib::detail, 90
read_response_line
 httpplib::ClientImpl, 273
read_socket
 httpplib::detail, 91
read_timeout_sec
 httpplib::ClientImpl, 303
 httpplib::detail::SocketStream, 508
 httpplib::Server, 493
read_timeout_usec
 httpplib::ClientImpl, 303
 httpplib::detail::SocketStream, 508
 httpplib::Server, 493
Reader
 httpplib::ContentReader, 309
reader_
 httpplib::ContentReader, 310
reason
 httpplib::Response, 410
redirect
 httpplib::ClientImpl, 274
 httpplib::detail, 92
redirect_count
 httpplib::Request, 394
regex_
 httpplib::detail::RegexMatcher, 380
RegexMatcher
 httpplib::detail::RegexMatcher, 380
release
 httpplib::detail::scope_exit, 430
remote_addr
 httpplib::Request, 394
remote_port
 httpplib::Request, 394
remove_mount_point
 httpplib::Server, 466
request_headers_
 httpplib::Result, 417
request_mutex_
 httpplib::ClientImpl, 304
RequestHeaderFieldsTooLarge_431
 httpplib, 19
RequestTimeout_408
 httpplib, 18
res_
 httpplib::Result, 417
ResetContent_205
 httpplib, 18

Response
 httpplib::Response, 397, 398
response_handler
 httpplib::Request, 394
ResponseHandler
 httpplib, 16
Result
 httpplib::Result, 412
ret_
 httpplib::detail::FileStat, 328
rotr
 anonymous_namespace{SHA256.cpp}, 9
routing
 httpplib::Server, 466
RSA, 417
 decrypt, 418
 encrypt, 419
 generate_keys, 420
 sign, 421
 verify, 422
RSA.cpp, 697, 700
 is_prime, 698
 modinv, 699
 random_bignum, 700
RSA.h, 674, 675
RSAPrivateKey, 424
 d, 425
 n, 425
RSAPublicKey, 426
 e, 427
 n, 427
saveKeys
 KeyStorage, 346
sb_
 httpplib::DataSink, 323
scope_exit
 httpplib::detail::scope_exit, 428, 429
SeeOther_303
 httpplib, 18
select_impl
 httpplib::detail, 93
select_read
 httpplib::detail, 94
select_write
 httpplib::detail, 95
send
 httpplib::Client, 194
 httpplib::ClientImpl, 276, 277
send_
 httpplib::ClientImpl, 278, 279
send_socket
 httpplib::detail, 96
send_with_content_provider
 httpplib::ClientImpl, 281, 282
separator
 httpplib::detail::PathParamsMatcher, 378
serialize_multipart_formdata
 httpplib::detail, 97

serialize_multipart_formdata_finish
 httpplib::detail, 98
serialize_multipart_formdata_get_content_type
 httpplib::detail, 99
serialize_multipart_formdata_item_begin
 httpplib::detail, 100
serialize_multipart_formdata_item_end
 httpplib::detail, 101
Server
 httpplib::Server, 437
ServiceUnavailable_503
 httpplib, 19
set_address_family
 httpplib::Client, 194
 httpplib::ClientImpl, 284
 httpplib::Server, 468
set_base_dir
 httpplib::Server, 469
set_basic_auth
 httpplib::Client, 195
 httpplib::ClientImpl, 284
set_bearer_token_auth
 httpplib::Client, 195
 httpplib::ClientImpl, 285
set_boundary
 httpplib::detail::MultipartFormDataParser, 366
set_chunked_content_provider
 httpplib::Response, 402
set_compress
 httpplib::Client, 195
 httpplib::ClientImpl, 285
set_connection_timeout
 httpplib::Client, 195
 httpplib::ClientImpl, 285
set_content
 httpplib::Response, 403, 404
set_content_provider
 httpplib::Response, 405
set_decompress
 httpplib::Client, 196
 httpplib::ClientImpl, 286
set_default_file_mimetype
 httpplib::Server, 469
set_default_headers
 httpplib::Client, 196
 httpplib::ClientImpl, 286
 httpplib::Server, 470
set_error_handler
 httpplib::Server, 470
set_error_handler_core
 httpplib::Server, 471
set_exception_handler
 httpplib::Server, 472
set_expect_100_continue_handler
 httpplib::Server, 472
set_file_content
 httpplib::Response, 406, 407
set_file_extension_and_mimetype_mapping
 httpplib::Server, 473
set_file_request_handler
 httpplib::Server, 473
set_follow_location
 httpplib::Client, 196
 httpplib::ClientImpl, 286
set_header
 httpplib::Request, 390
 httpplib::Response, 407
set_header_writer
 httpplib::Client, 196
 httpplib::ClientImpl, 286
 httpplib::Server, 473
set_hostname_addr_map
 httpplib::Client, 196
 httpplib::ClientImpl, 287
set_idle_interval
 httpplib::Server, 474
set_interface
 httpplib::Client, 197
 httpplib::ClientImpl, 287
set_ipv6_v6only
 httpplib::ClientImpl, 287
 httpplib::Server, 475
set_keep_alive
 httpplib::Client, 197
 httpplib::ClientImpl, 287
set_keep_alive_max_count
 httpplib::Server, 476
set_keep_alive_timeout
 httpplib::Server, 476
set_logger
 httpplib::Client, 197
 httpplib::ClientImpl, 287
 httpplib::Server, 477
set_max_timeout
 httpplib::Client, 197
 httpplib::ClientImpl, 287, 288
set_mount_point
 httpplib::Server, 477
set_nonblocking
 httpplib::detail, 102
set_payload_max_length
 httpplib::Server, 478
set_post_routing_handler
 httpplib::Server, 478
set_pre_routing_handler
 httpplib::Server, 479
set_proxy
 httpplib::Client, 198
 httpplib::ClientImpl, 288
set_proxy_basic_auth
 httpplib::Client, 198
 httpplib::ClientImpl, 289
set_proxy_bearer_token_auth
 httpplib::Client, 198
 httpplib::ClientImpl, 289
set_read_timeout

httpplib::Client, 199
httpplib::ClientImpl, 289, 290
httpplib::Server, 479, 480
set_redirect
 httpplib::Response, 408
set_socket_opt
 httpplib::detail, 103
set_socket_opt_impl
 httpplib::detail, 103
set_socket_opt_time
 httpplib::detail, 104
set_socket_options
 httpplib::Client, 199
 httpplib::ClientImpl, 290
 httpplib::Server, 481
set_tcp_nodelay
 httpplib::Client, 199
 httpplib::ClientImpl, 291
 httpplib::Server, 481
set_url_encode
 httpplib::Client, 199
 httpplib::ClientImpl, 291
set_write_timeout
 httpplib::Client, 200
 httpplib::ClientImpl, 291
 httpplib::Server, 482
SHA256, 495
 hash, 495
SHA256.cpp, 702, 704
SHA256.h, 676
shutdown
 httpplib::TaskQueue, 523
 httpplib::ThreadPool, 526
shutdown_
 httpplib::ThreadPool, 527
shutdown_socket
 httpplib::ClientImpl, 292
 httpplib::detail, 105
shutdown_ssl
 httpplib::ClientImpl, 293
sign
 RSA, 421
sink_
 httpplib::DataSink::data_sink_streambuf, 313
size
 httpplib::detail::mmap, 354
 httpplib::detail::stream_line_reader, 519
size_
 httpplib::detail::mmap, 355
skip_content_with_length
 httpplib::detail, 106
small_sigma0
 anonymous_namespace{SHA256.cpp}, 10
small_sigma1
 anonymous_namespace{SHA256.cpp}, 11
sock
 httpplib::ClientImpl::Socket, 498
sock_
 httpplib::detail::SocketStream, 508
socket
 httpplib::Client, 200
 httpplib::ClientImpl, 294
 httpplib::detail::BufferStream, 165
 httpplib::detail::SocketStream, 505
 httpplib::Stream, 512
socket_
 httpplib::ClientImpl, 304
socket_mutex_
 httpplib::ClientImpl, 304
socket_options_
 httpplib::ClientImpl, 304
 httpplib::Server, 494
socket_requests_are_from_thread_
 httpplib::ClientImpl, 304
socket_requests_in_flight_
 httpplib::ClientImpl, 304
socket_should_be_closed_when_request_is_done_
 httpplib::ClientImpl, 304
socket_t
 httpplib.h, 549
SocketOptions
 httpplib, 16
SocketStream
 httpplib::detail::SocketStream, 502
split
 httpplib::detail, 107, 108
SSLConnection
 httpplib, 17
SSLLoadingCerts
 httpplib, 17
SSLPeerCouldBeClosed_
 httpplib, 17
SSLServerHostnameVerification
 httpplib, 17
SSLServerVerification
 httpplib, 17
SSLVerifierResponse
 httpplib, 17
st_
 httpplib::detail::FileStat, 328
start
 HttpServer, 331
start_time_
 httpplib::detail::SocketStream, 508
 httpplib::Request, 394
start_with
 httpplib::detail::MultipartFormDataParser, 366
start_with_case_ignore
 httpplib::detail::MultipartFormDataParser, 367
state_
 httpplib::detail::MultipartFormDataParser, 369
static_fragments
 httpplib::detail::PathParamsMatcher, 378
status
 httpplib::Response, 410
status_message

httpplib, 26
StatusCode
 httpplib, 17
stop
 httpplib::Client, 200
 httpplib::ClientImpl, 295
 httpplib::Server, 483
str2tag
 httpplib::detail, 108
str2tag_core
 httpplib::detail, 109
str_len
 httpplib::detail, 110
stream_line_reader
 httpplib::detail::stream_line_reader, 517
strm_—
 httpplib::detail::stream_line_reader, 520
Success
 httpplib, 17
svr_sock_—
 httpplib::Server, 494
SwitchingProtocol_101
 httpplib, 18

target
 httpplib::Request, 394
TaskQueue
 httpplib::TaskQueue, 522
tcp_nodelay_—
 httpplib::ClientImpl, 304
 httpplib::Server, 494
TemporaryRedirect_307
 httpplib, 18
ThreadPool
 httpplib::ThreadPool, 525, 526
threads_—
 httpplib::ThreadPool, 528
to_lower
 httpplib::detail::case_ignore, 125
to_string
 httpplib, 27
to_uint32
 anonymous_namespace{SHA256.cpp}, 11
to_utf8
 httpplib::detail, 110
TooEarly_425
 httpplib, 19
TooManyRequests_429
 httpplib, 19
toString
 BigInt, 157, 158
trim
 BigInt, 159
 httpplib::detail, 111
trim_copy
 httpplib::detail, 112
trim_double_quotes_copy
 httpplib::detail, 112

 Unauthorized_401
 httpplib, 18
UnavailableForLegalReasons_451
 httpplib, 19
unescape_abstract_namespace_unix_domain
 httpplib::detail, 113
Unhandled
 httpplib::Server, 437
Unknown
 httpplib, 17
UnprocessableContent_422
 httpplib, 19
UnsupportedMediaType_415
 httpplib, 18
UnsupportedMultipartBoundaryChars
 httpplib, 17
unused_306
 httpplib, 18
UpgradeRequired_426
 httpplib, 19
UriTooLong_414
 httpplib, 18
url_encode_—
 httpplib::ClientImpl, 305
USE_IF2IP
 httpplib.h, 549
UseProxy_305
 httpplib, 18
User, 528
 id, 529
 password, 529
 username, 529
username
 User, 529

value
 httpplib::Result, 416
VariantAlsoNegotiates_506
 httpplib, 19
verify
 RSA, 422
verifyAccessToken
 JWT, 340
verifyRefreshToken
 JWT, 342
version
 httpplib::Request, 394
 httpplib::Response, 410

wait_readable
 httpplib::detail::BufferStream, 165
 httpplib::detail::SocketStream, 505
 httpplib::Stream, 513
wait_until_ready
 httpplib::Server, 483
wait_until_socket_is_ready
 httpplib::detail, 113
wait_writable
 httpplib::detail::BufferStream, 165

httpplib::detail::SocketStream, 506
 httpplib::Stream, 513
worker
 httpplib::ThreadPool, 527
 httpplib::ThreadPool::worker, 531
Write
 httpplib, 17
write
 httpplib::DataSink, 323
 httpplib::detail::BufferStream, 165
 httpplib::detail::SocketStream, 507
 httpplib::Stream, 513, 514
write_content
 httpplib::detail, 115
write_content_chunked
 httpplib::detail, 117
write_content_with_provider
 httpplib::ClientImpl, 296
 httpplib::Server, 483
write_content_without_length
 httpplib::detail, 119
write_data
 httpplib::detail, 120
write_headers
 httpplib::detail, 121
write_multipart_ranges_data
 httpplib::detail, 122
write_request
 httpplib::ClientImpl, 297
write_request_line
 httpplib::detail, 123
write_response
 httpplib::Server, 485
write_response_core
 httpplib::Server, 486
write_response_line
 httpplib::detail, 124
write_response_with_content
 httpplib::Server, 488
write_timeout_sec_
 httpplib::ClientImpl, 305
 httpplib::detail::SocketStream, 508
 httpplib::Server, 494
write_timeout_usec_
 httpplib::ClientImpl, 305
 httpplib::detail::SocketStream, 508
 httpplib::Server, 494
xspputn
 httpplib::DataSink::data_sink_streambuf, 313

Zstd
 httpplib::detail, 32