

SOIL C++

Generated by Doxygen 1.9.3



<b>1 SOIL C++</b>	<b>1</b>
1.1 Development setup	1
1.1.1 Dependencies	1
1.1.2 Windows	2
1.1.3 Linux	2
1.2 Implement device conform to the unified interface	2
1.3 Known Issues and Warnings	3
<b>2 Todo List</b>	<b>5</b>
<b>3 Namespace Index</b>	<b>7</b>
3.1 Namespace List	7
<b>4 Hierarchical Index</b>	<b>9</b>
4.1 Class Hierarchy	9
<b>5 Class Index</b>	<b>11</b>
5.1 Class List	11
<b>6 File Index</b>	<b>13</b>
6.1 File List	13
<b>7 Namespace Documentation</b>	<b>15</b>
7.1 HTTP Namespace Reference	15
7.1.1 Typedef Documentation	15
7.1.1.1 Json	15
7.1.1.2 Methods	16
7.1.1.3 Request	16
7.1.1.4 Response	16
7.1.1.5 Status	16
7.2 MQTT Namespace Reference	16
7.3 mqtt Namespace Reference	17
7.4 SIGN Namespace Reference	17
7.5 SOIL Namespace Reference	17
7.5.1 Detailed Description	19
7.5.2 Typedef Documentation	19
7.5.2.1 BOOL	19
7.5.2.2 DIMENSION	19
7.5.2.3 DOUBLE	19
7.5.2.4 ENUM	19
7.5.2.5 INT	20
7.5.2.6 STRING	20
7.5.2.7 TIME	20
7.5.3 Function Documentation	20

7.5.3.1 datatype()	20
7.5.3.2 datatype< bool >()	21
7.5.3.3 datatype< double >()	21
7.5.3.4 datatype< int >()	21
7.5.3.5 datatype< int64_t >()	21
7.5.3.6 datatype< SOIL::ENUM >()	21
7.5.3.7 datatype< SOIL::TIME >()	22
7.5.3.8 datatype< std::string >()	22
7.5.3.9 null_deleter()	22
7.5.3.10 operator<=>()	22
7.5.3.11 operator>=()	23
7.5.3.12 to_json()	23
7.5.3.13 to_json< double >()	24
7.5.3.14 to_json< int >()	24
7.5.3.15 to_json< int64_t >()	24
7.5.3.16 to_json< SOIL::BOOL >()	24
7.5.3.17 to_json< SOIL::ENUM >()	24
7.5.3.18 to_json< SOIL::TIME >()	25
7.5.3.19 to_json< std::string >()	25
7.5.3.20 to_value()	25
7.5.3.21 to_value< bool >()	25
7.5.3.22 to_value< double >()	26
7.5.3.23 to_value< int >()	26
7.5.3.24 to_value< int64_t >()	26
7.5.3.25 to_value< SOIL::ENUM >()	26
7.5.3.26 to_value< SOIL::TIME >()	26
7.5.3.27 to_value< std::string >()	26
7.6 UDP Namespace Reference	27
<b>8 Class Documentation</b>	<b>29</b>
8.1 UDP::Broadcast Class Reference	29
8.1.1 Detailed Description	29
8.1.2 Constructor & Destructor Documentation	30
8.1.2.1 Broadcast()	30
8.1.2.2 ~Broadcast()	30
8.1.3 Member Function Documentation	30
8.1.3.1 add_client()	30
8.1.3.2 configure()	31
8.1.3.3 remove_client()	31
8.1.3.4 send() [1/2]	31
8.1.3.5 send() [2/2]	32
8.1.3.6 use_count()	32

8.2 MQTT::Configuration Class Reference . . . . .	32
8.2.1 Detailed Description . . . . .	33
8.2.2 Constructor & Destructor Documentation . . . . .	33
8.2.2.1 Configuration() [1/2] . . . . .	33
8.2.2.2 Configuration() [2/2] . . . . .	34
8.2.2.3 ~Configuration() . . . . .	34
8.2.3 Member Function Documentation . . . . .	34
8.2.3.1 uri() . . . . .	34
8.2.4 Member Data Documentation . . . . .	34
8.2.4.1 certificate_authority . . . . .	35
8.2.4.2 clean_session . . . . .	35
8.2.4.3 connection_timeout_s . . . . .	35
8.2.4.4 host . . . . .	35
8.2.4.5 keep_alive . . . . .	36
8.2.4.6 min_delay_ms . . . . .	36
8.2.4.7 password . . . . .	36
8.2.4.8 path . . . . .	36
8.2.4.9 port . . . . .	36
8.2.4.10 root . . . . .	37
8.2.4.11 ssl . . . . .	37
8.2.4.12 username . . . . .	37
8.2.4.13 verify . . . . .	37
8.2.4.14 websocket . . . . .	38
8.3 UDP::Configuration Class Reference . . . . .	38
8.3.1 Detailed Description . . . . .	38
8.3.2 Member Typedef Documentation . . . . .	39
8.3.2.1 iterator . . . . .	39
8.3.3 Constructor & Destructor Documentation . . . . .	39
8.3.3.1 Configuration() [1/2] . . . . .	39
8.3.3.2 Configuration() [2/2] . . . . .	39
8.3.3.3 ~Configuration() . . . . .	39
8.3.4 Member Data Documentation . . . . .	40
8.3.4.1 clients . . . . .	40
8.4 SOIL::Container< T, x, y > Class Template Reference . . . . .	40
8.4.1 Detailed Description . . . . .	41
8.4.2 Constructor & Destructor Documentation . . . . .	42
8.4.2.1 Container() [1/3] . . . . .	42
8.4.2.2 Container() [2/3] . . . . .	42
8.4.2.3 Container() [3/3] . . . . .	42
8.4.3 Member Function Documentation . . . . .	43
8.4.3.1 at() . . . . .	43
8.4.3.2 check_range() . . . . .	43

8.4.3.3 is_null()	44
8.4.3.4 operator*()	44
8.4.3.5 serialize_dimensions()	44
8.4.3.6 serialize_value()	45
8.4.3.7 set_null()	45
8.4.3.8 wjson()	45
8.5 SOIL::Container< T, -1, -1 > Class Template Reference	46
8.5.1 Detailed Description	46
8.5.2 Constructor & Destructor Documentation	46
8.5.2.1 Container() [1/3]	46
8.5.2.2 Container() [2/3]	47
8.5.2.3 Container() [3/3]	47
8.5.3 Member Function Documentation	47
8.5.3.1 at()	47
8.5.3.2 check_range()	47
8.5.3.3 is_null()	47
8.5.3.4 operator*()	48
8.5.3.5 serialize_dimensions()	48
8.5.3.6 serialize_value()	48
8.5.3.7 set_null()	48
8.5.3.8 wjson()	48
8.6 SOIL::Container< T, x, -1 > Class Template Reference	48
8.6.1 Detailed Description	49
8.6.2 Constructor & Destructor Documentation	49
8.6.2.1 Container() [1/3]	49
8.6.2.2 Container() [2/3]	49
8.6.2.3 Container() [3/3]	50
8.6.3 Member Function Documentation	50
8.6.3.1 at()	50
8.6.3.2 check_range()	50
8.6.3.3 is_null()	50
8.6.3.4 operator*()	50
8.6.3.5 serialize_dimensions()	51
8.6.3.6 serialize_value()	51
8.6.3.7 set_null()	51
8.6.3.8 wjson()	51
8.7 SOIL::Element Class Reference	51
8.7.1 Detailed Description	53
8.7.2 Constructor & Destructor Documentation	53
8.7.2.1 Element()	53
8.7.2.2 ~Element()	53
8.7.3 Member Function Documentation	54

8.7.3.1 add() [1/2]	54
8.7.3.2 add() [2/2]	54
8.7.3.3 cast()	55
8.7.3.4 fqid()	55
8.7.3.5 handle()	55
8.7.3.6 insert() [1/2]	56
8.7.3.7 insert() [2/2]	56
8.7.3.8 is_function()	57
8.7.3.9 is_object()	57
8.7.3.10 is_parameter()	58
8.7.3.11 is_variable()	58
8.7.3.12 json()	58
8.7.3.13 operator[]()	58
8.7.3.14 remove()	59
8.7.3.15 wjson()	59
8.7.4 Member Data Documentation	60
8.7.4.1 children	60
8.7.4.2 description	60
8.7.4.3 mutex	60
8.7.4.4 name	60
8.7.4.5 ontology	61
8.7.4.6 parent	61
8.7.4.7 self	61
8.7.4.8 uuid	61
8.8 SOIL::Enum Class Reference	62
8.8.1 Detailed Description	62
8.8.2 Constructor & Destructor Documentation	62
8.8.2.1 Enum() [1/3]	62
8.8.2.2 Enum() [2/3]	62
8.8.2.3 Enum() [3/3]	63
8.8.2.4 ~Enum()	63
8.8.3 Member Function Documentation	63
8.8.3.1 choices()	63
8.8.3.2 index() [1/2]	64
8.8.3.3 index() [2/2]	64
8.8.3.4 selected()	64
8.8.3.5 set() [1/2]	65
8.8.3.6 set() [2/2]	65
8.9 MQTT::Exception Class Reference	65
8.9.1 Detailed Description	66
8.9.2 Constructor & Destructor Documentation	66
8.9.2.1 Exception() [1/2]	66

8.9.2.2 Exception() [2/2]	67
8.9.2.3 ~Exception()	67
8.9.3 Member Function Documentation	67
8.9.3.1 code()	67
8.10 UDP::Exception Class Reference	68
8.10.1 Detailed Description	68
8.10.2 Constructor & Destructor Documentation	68
8.10.2.1 Exception()	68
8.10.2.2 ~Exception()	69
8.11 SOIL::Figure< T, x, y > Class Template Reference	69
8.11.1 Detailed Description	70
8.11.2 Constructor & Destructor Documentation	70
8.11.2.1 Figure()	70
8.11.2.2 ~Figure()	71
8.11.3 Member Function Documentation	71
8.11.3.1 cast()	71
8.11.3.2 check_range()	72
8.11.3.3 operator*()	72
8.11.3.4 operator=()	72
8.11.3.5 read()	73
8.11.3.6 set_range()	73
8.11.3.7 set_time()	73
8.11.3.8 set_value()	74
8.11.3.9 update()	74
8.11.3.10 wjson()	75
8.11.3.11 write()	75
8.11.4 Member Data Documentation	75
8.11.4.1 range	75
8.11.4.2 time	76
8.11.4.3 unit	76
8.11.4.4 value	76
8.12 SOIL::Function Class Reference	76
8.12.1 Detailed Description	77
8.12.2 Constructor & Destructor Documentation	78
8.12.2.1 Function()	78
8.12.2.2 ~Function()	78
8.12.3 Member Function Documentation	78
8.12.3.1 add_argument() [1/2]	78
8.12.3.2 add_argument() [2/2]	79
8.12.3.3 add_return()	80
8.12.3.4 create()	80
8.12.3.5 handle_get()	81



8.12.3.6 handle_post()	81
8.12.3.7 invoke()	82
8.12.3.8 make_argument()	82
8.12.3.9 make_return()	83
8.12.3.10 ptr()	84
8.12.3.11 wjson()	84
8.13 SIGN::Hasher Class Reference	84
8.13.1 Detailed Description	85
8.13.2 Constructor & Destructor Documentation	85
8.13.2.1 Hasher()	85
8.13.2.2 ~Hasher()	85
8.13.3 Member Function Documentation	85
8.13.3.1 hash()	86
8.13.3.2 print()	86
8.13.3.3 push_back()	86
8.13.3.4 reset()	87
8.13.3.5 sha256()	87
8.13.3.6 size()	87
8.14 MQTT::MessageContainer Struct Reference	88
8.14.1 Detailed Description	88
8.14.2 Member Data Documentation	88
8.14.2.1 message	88
8.14.2.2 qos	89
8.14.2.3 retain	89
8.14.2.4 topic	89
8.15 SOIL::Object Class Reference	89
8.15.1 Detailed Description	90
8.15.2 Constructor & Destructor Documentation	90
8.15.2.1 Object()	91
8.15.2.2 ~Object()	91
8.15.3 Member Function Documentation	91
8.15.3.1 create()	91
8.15.3.2 handle_delete()	92
8.15.3.3 handle_get()	92
8.15.3.4 handle_put()	93
8.15.3.5 insert()	93
8.15.3.6 ptr()	94
8.15.3.7 read()	94
8.15.3.8 remove()	95
8.15.3.9 wjson()	95
8.16 SOIL::Parameter< T, x, y > Class Template Reference	95
8.16.1 Detailed Description	97

8.16.2 Constructor & Destructor Documentation	97
8.16.2.1 Parameter()	97
8.16.2.2 ~Parameter()	98
8.16.3 Member Function Documentation	98
8.16.3.1 create()	98
8.16.3.2 handle_get()	99
8.16.3.3 handle_patch()	99
8.16.3.4 mqtt()	100
8.16.3.5 operator=()	101
8.16.3.6 ptr()	101
8.16.3.7 read()	101
8.16.3.8 wjson()	102
8.16.3.9 write()	102
8.16.4 Member Data Documentation	102
8.16.4.1 constant	102
8.17 MQTT::Publisher Class Reference	103
8.17.1 Detailed Description	103
8.17.2 Constructor & Destructor Documentation	104
8.17.2.1 Publisher()	104
8.17.2.2 ~Publisher()	104
8.17.3 Member Function Documentation	104
8.17.3.1 configure()	104
8.17.3.2 connect() [1/2]	105
8.17.3.3 connect() [2/2]	105
8.17.3.4 disconnect()	105
8.17.3.5 is_connected()	105
8.17.3.6 min_delay_ms()	106
8.17.3.7 publish() [1/2]	106
8.17.3.8 publish() [2/2]	107
8.17.3.9 reconnect()	107
8.17.3.10 set_buffer()	107
8.17.3.11 set_root_topic()	108
8.18 SOIL::Range< T > Class Template Reference	108
8.18.1 Detailed Description	108
8.18.2 Constructor & Destructor Documentation	109
8.18.2.1 Range() [1/3]	109
8.18.2.2 Range() [2/3]	109
8.18.2.3 Range() [3/3]	109
8.18.2.4 ~Range()	110
8.18.3 Member Function Documentation	110
8.18.3.1 check()	110
8.18.3.2 wjson()	110

8.19 SOIL::Range< ENUM > Class Reference	111
8.19.1 Detailed Description	111
8.19.2 Constructor & Destructor Documentation	111
8.19.2.1 Range() [1/2]	111
8.19.2.2 Range() [2/2]	111
8.19.2.3 ~Range()	112
8.19.3 Member Function Documentation	112
8.19.3.1 check() [1/2]	112
8.19.3.2 check() [2/2]	112
8.19.3.3 wjson()	112
8.20 SOIL::Range< std::string > Class Reference	112
8.20.1 Detailed Description	113
8.20.2 Constructor & Destructor Documentation	113
8.20.2.1 Range() [1/3]	113
8.20.2.2 Range() [2/3]	113
8.20.2.3 Range() [3/3]	113
8.20.2.4 ~Range()	113
8.20.3 Member Function Documentation	113
8.20.3.1 check()	114
8.20.3.2 wjson()	114
8.21 HTTP::Resource Class Reference	114
8.21.1 Detailed Description	115
8.21.2 Constructor & Destructor Documentation	115
8.21.2.1 Resource()	116
8.21.2.2 ~Resource()	116
8.21.3 Member Function Documentation	116
8.21.3.1 apply_headers()	116
8.21.3.2 handle()	116
8.21.3.3 handle_delete()	117
8.21.3.4 handle_exception()	117
8.21.3.5 handle_get()	118
8.21.3.6 handle_head()	118
8.21.3.7 handle_options()	119
8.21.3.8 handle_patch()	119
8.21.3.9 handle_post()	120
8.21.3.10 handle_put()	120
8.21.3.11 request_info()	121
8.21.4 Member Data Documentation	121
8.21.4.1 allowed_methods	121
8.21.4.2 allowed_origins	121
8.21.4.3 content_type	122
8.22 HTTP::Server Class Reference	122

8.22.1 Detailed Description	122
8.22.2 Constructor & Destructor Documentation	122
8.22.2.1 Server()	122
8.22.2.2 ~Server()	123
8.22.3 Member Function Documentation	123
8.22.3.1 add()	123
8.22.3.2 close()	124
8.22.3.3 open()	124
8.22.3.4 remove()	124
8.23 SIGN::Signer Class Reference	124
8.23.1 Detailed Description	125
8.23.2 Constructor & Destructor Documentation	125
8.23.2.1 Signer()	125
8.23.2.2 ~Signer()	125
8.23.3 Member Function Documentation	126
8.23.3.1 name()	126
8.23.3.2 openssl_version()	126
8.23.3.3 sign()	126
8.24 SOIL::Time Class Reference	127
8.24.1 Detailed Description	128
8.24.2 Constructor & Destructor Documentation	128
8.24.2.1 Time() [1/3]	128
8.24.2.2 Time() [2/3]	128
8.24.2.3 Time() [3/3]	128
8.24.2.4 ~Time()	129
8.24.3 Member Function Documentation	129
8.24.3.1 is_null()	129
8.24.3.2 rfc3339()	130
8.24.3.3 serialize()	130
8.24.3.4 set_null()	130
8.24.3.5 utc_now()	131
8.24.4 Friends And Related Function Documentation	131
8.24.4.1 operator<=	131
8.24.4.2 operator>=	132
8.25 SOIL::Variable< T, x, y > Class Template Reference	132
8.25.1 Detailed Description	134
8.25.2 Constructor & Destructor Documentation	134
8.25.2.1 Variable()	135
8.25.2.2 ~Variable()	135
8.25.3 Member Function Documentation	135
8.25.3.1 bytes()	136
8.25.3.2 create()	136

8.25.3.3 fingerprint()	137
8.25.3.4 handle_get()	137
8.25.3.5 handle_options()	138
8.25.3.6 mqtt()	138
8.25.3.7 operator=()	139
8.25.3.8 ptr()	139
8.25.3.9 read()	140
8.25.3.10 set_covariance()	140
8.25.3.11 sha256()	140
8.25.3.12 sign()	141
8.25.3.13 update()	141
8.25.3.14 wjson()	142
8.25.3.15 write()	142
8.25.4 Member Data Documentation	142
8.25.4.1 covariance	143
8.25.4.2 hash	143
8.25.4.3 nonce	143
<b>9 File Documentation</b>	<b>145</b>
9.1 README.md File Reference	145
9.2 src/MQTT/Configuration.cpp File Reference	145
9.2.1 Typedef Documentation	145
9.2.1.1 json	145
9.3 Configuration.cpp	146
9.4 src/UDP/Configuration.cpp File Reference	147
9.4.1 Typedef Documentation	147
9.4.1.1 json	147
9.5 Configuration.cpp	147
9.6 src/MQTT/Configuration.h File Reference	148
9.7 Configuration.h	148
9.8 src/UDP/Configuration.h File Reference	149
9.9 Configuration.h	149
9.10 src/MQTT/constants.h File Reference	149
9.10.1 Macro Definition Documentation	149
9.10.1.1 _WIN32_WINNT	150
9.11 constants.h	150
9.12 src/REST/constants.h File Reference	150
9.12.1 Macro Definition Documentation	150
9.12.1.1 _WIN32_WINNT	150
9.13 constants.h	151
9.14 src/SIGN/constants.h File Reference	151
9.14.1 Macro Definition Documentation	151

9.14.1.1 _WIN32_WINNT . . . . .	151
9.15 constants.h . . . . .	151
9.16 src/SOIL/constants.h File Reference . . . . .	152
9.16.1 Macro Definition Documentation . . . . .	152
9.16.1.1 _WIN32_WINNT . . . . .	152
9.17 constants.h . . . . .	152
9.18 src/UDP/constants.h File Reference . . . . .	152
9.19 constants.h . . . . .	152
9.20 src/MQTT/LocalException.cpp File Reference . . . . .	153
9.21 LocalException.cpp . . . . .	153
9.22 src/MQTT/LocalException.h File Reference . . . . .	153
9.23 LocalException.h . . . . .	153
9.24 src/MQTT/MessageContainer.h File Reference . . . . .	154
9.25 MessageContainer.h . . . . .	154
9.26 src/MQTT/Publisher.cpp File Reference . . . . .	154
9.26.1 Macro Definition Documentation . . . . .	155
9.26.1.1 PAHO_MQTT_IMPORTS . . . . .	155
9.27 Publisher.cpp . . . . .	155
9.28 src/MQTT/Publisher.h File Reference . . . . .	158
9.29 Publisher.h . . . . .	158
9.30 src/REST/Resource.cpp File Reference . . . . .	159
9.31 Resource.cpp . . . . .	160
9.32 src/REST/Resource.h File Reference . . . . .	162
9.33 Resource.h . . . . .	162
9.34 src/REST/Server.cpp File Reference . . . . .	163
9.35 Server.cpp . . . . .	163
9.36 src/REST/Server.h File Reference . . . . .	164
9.37 Server.h . . . . .	164
9.38 src/REST/Types.h File Reference . . . . .	164
9.39 Types.h . . . . .	165
9.40 src/SOIL/Types.h File Reference . . . . .	165
9.41 Types.h . . . . .	166
9.42 src/SIGN/Hasher.cpp File Reference . . . . .	166
9.43 Hasher.cpp . . . . .	167
9.44 src/SIGN/Hasher.h File Reference . . . . .	167
9.45 Hasher.h . . . . .	168
9.46 src/SIGN/Signer.cpp File Reference . . . . .	168
9.47 Signer.cpp . . . . .	168
9.48 src/SIGN/Signer.h File Reference . . . . .	169
9.48.1 Typedef Documentation . . . . .	170
9.48.1.1 EVP_MD_CTX . . . . .	170
9.48.1.2 EVP_PKEY . . . . .	170

9.49 Signer.h . . . . .	170
9.50 src/SOIL/Container.cpp File Reference . . . . .	171
9.51 Container.cpp . . . . .	171
9.52 src/SOIL/Container.h File Reference . . . . .	171
9.53 Container.h . . . . .	171
9.54 src/SOIL/Element.cpp File Reference . . . . .	177
9.55 Element.cpp . . . . .	177
9.56 src/SOIL/Element.h File Reference . . . . .	179
9.57 Element.h . . . . .	180
9.58 src/SOIL/Enum.cpp File Reference . . . . .	181
9.59 Enum.cpp . . . . .	181
9.60 src/SOIL/Enum.h File Reference . . . . .	182
9.61 Enum.h . . . . .	183
9.62 src/SOIL/Figure.cpp File Reference . . . . .	183
9.62.1 Function Documentation . . . . .	183
9.62.1.1 SOIL::datatype< SOIL::ENUM >() . . . . .	183
9.62.1.2 SOIL::datatype< SOIL::TIME >() . . . . .	184
9.62.1.3 SOIL::datatype< std::string >() . . . . .	184
9.63 Figure.cpp . . . . .	184
9.64 src/SOIL/Figure.h File Reference . . . . .	185
9.65 Figure.h . . . . .	185
9.66 src/SOIL/Function.cpp File Reference . . . . .	187
9.67 Function.cpp . . . . .	188
9.68 src/SOIL/Function.h File Reference . . . . .	189
9.69 Function.h . . . . .	189
9.70 src/SOIL/json_helpers.cpp File Reference . . . . .	191
9.70.1 Function Documentation . . . . .	191
9.70.1.1 SOIL::to_json< SOIL::BOOL >() . . . . .	191
9.70.1.2 SOIL::to_json< SOIL::ENUM >() . . . . .	191
9.70.1.3 SOIL::to_json< SOIL::TIME >() . . . . .	191
9.70.1.4 SOIL::to_json< std::string >() . . . . .	192
9.70.1.5 SOIL::to_value< SOIL::ENUM >() . . . . .	192
9.70.1.6 SOIL::to_value< SOIL::TIME >() . . . . .	192
9.70.1.7 SOIL::to_value< std::string >() . . . . .	192
9.71 json_helpers.cpp . . . . .	193
9.72 src/SOIL/json_helpers.h File Reference . . . . .	194
9.73 json_helpers.h . . . . .	195
9.74 src/SOIL/Object.cpp File Reference . . . . .	195
9.75 Object.cpp . . . . .	195
9.76 src/SOIL/Object.h File Reference . . . . .	196
9.77 Object.h . . . . .	197
9.78 src/SOIL/Parameter.cpp File Reference . . . . .	197

9.79 Parameter.cpp . . . . .	197
9.80 src/SOIL/Parameter.h File Reference . . . . .	198
9.81 Parameter.h . . . . .	198
9.82 src/SOIL/Range.cpp File Reference . . . . .	200
9.83 Range.cpp . . . . .	200
9.84 src/SOIL/Range.h File Reference . . . . .	202
9.85 Range.h . . . . .	202
9.86 src/SOIL/Time.cpp File Reference . . . . .	204
9.87 Time.cpp . . . . .	204
9.88 src/SOIL/Time.h File Reference . . . . .	206
9.89 Time.h . . . . .	206
9.90 src/SOIL/Types.cpp File Reference . . . . .	207
9.91 Types.cpp . . . . .	207
9.92 src/SOIL/Variable.cpp File Reference . . . . .	207
9.93 Variable.cpp . . . . .	207
9.94 src/SOIL/Variable.h File Reference . . . . .	207
9.95 Variable.h . . . . .	208
9.96 src/TEST/main.cpp File Reference . . . . .	211
9.96.1 Function Documentation . . . . .	211
9.96.1.1 main() . . . . .	211
9.97 main.cpp . . . . .	211
9.98 src/UDP/Broadcast.cpp File Reference . . . . .	213
9.99 Broadcast.cpp . . . . .	213
9.100 src/UDP/Broadcast.h File Reference . . . . .	215
9.101 Broadcast.h . . . . .	215
9.102 src/UDP/Exception.cpp File Reference . . . . .	216
9.103 Exception.cpp . . . . .	216
9.104 src/UDP/Exception.h File Reference . . . . .	216
9.105 Exception.h . . . . .	217

<b>Index</b>	<b>219</b>
--------------	------------



# Chapter 1

## SOIL C++

The **Unified Device Interface** provides four dynamic libraries which assemble a web-service for a sensor or measurement device. The interface provides a RESTful [HTTP](#) server, a MQTT-Publisher, and UDP-Broadcaster, whereas the UDP-Broadcaster is of minor importance. The library allows to fully implement a [SOIL](#) sensor service. By implementing your device against the provided API, it is possible to send message via [MQTT](#) which are then recieved by clients and to access the measurements and methods of the sensor/device via [HTTP](#).

If a more detailed introduction or explanation of these concepts is requiried, please contact [Benjamin Montavon](#) or [Matthias Bodenbenner](#). Same holds for any kind of problems when using, compiling, etc. the libraries.

### 1.1 Development setup

#### 1.1.1 Dependencies

This library has the following dependencies (VCPKG naming):

- boost-date-time
- boost-asio
- boost-thread
- boost-algorithm
- nlohmann-json
- cpprestsdk
- paho-mqttpp3
- openssl

### 1.1.2 Windows

1. Install Visual Studio - The most recent developments have been carried out with Visual Studio 2022. The Community Edition is known to be sufficient and can be used free of charge for academic purposes. You can download it and see more details at <https://visualstudio.microsoft.com/de/vs/community/>.
2. Install VCPKG
  - (a) Clone the [vcpkg-repository](#).
  - (b) Go into vcpkg folder.
  - (c) Run `*.\bootstrap-vcpkg.bat*`
  - (a) Configure environment variables and path:
    - i. Add environment variable `VCPKG_INSTALLED` and set the value to `[vcpkg-directory]/installed`. The Visual Studio project heavily depends on this path!
    - ii. Add the vcpkg directory to `PATH`.
  - (b) Do not forget to restart Visual Studio after changing the variables!
3. Install required packages. You can install **all** dependencies by executing `install_packages.bat`. It may take a lot of time (> 1h), have patience!. If you only want to use a specific library install the following packages. For compiling the x64 configuration install the packages and append `*:x64-windows*` (e.g. `boost-date-time:x64-windows`).
4. Compile source code.
5. The compiled `*.dll`-files and generated `*.lib` are copied into the `dist` folder under the chosen configuration (i.e. `./x64/Debug`)
6. The `*.dll` files coming with the library still reside within `VCPKG_INSTALLED` within the respective subfolders (`<x64/x86>-windows/[debug/]bin`). Adding all folders to `PATH` may cause problems during debugging. It is recommended to put them next to the application or carefully select path modifications. During debugging, Visual Studio is managing this problem for you.

### 1.1.3 Linux

1. Install `git`, `gcc`, `make`, `cmake` at least and configure your build environment
2. Install VCPKG
  - (a) Clone the [vcpkg-repository](#).
  - (b) Go into vcpkg folder.
  - (c) Run `*./bootstrap-vcpkg.sh*`
3. Install required packages. You may use `install_packages.sh`.
4. Run CMAKE using the vcpkg toolchain according to <https://vcpkg.readthedocs.io/en/latest/examples/installing-and-using-packages/#cmake>. When using the CMake GUI, use the option *Specify Toolchain for Cross-Compiling*, even if not Cross-Compiling. From the command-line, use `CMAKE_TOOLCHAIN_FILE`. Using `projects/linux/cmake/CMakeLists.txt` will build all components and test at once.

## 1.2 Implement device conform to the unified interface

To implement a device service which is conform to the unified interface one has to inherit from the three classes *Object*, *Function* and *Variable* of the **REST** project. The `main.cpp` of the *TEST* project can be used as template for main method of your device.

## 1.3 Known Issues and Warnings

This project recently changed the default string backend for `cpprestsdk` from `std::wstring` to `std::string`. You may need to adjust some little pieces of code at your end when updating.



## Chapter 2

# Todo List

### Class `HTTP::Server`

cpprestsdk is now in maintenance mode, if this library shall be in the very long term, a substitute may need to be found.

### Class `MQTT::Publisher`

Implement support for authenticating with client certificates.

### Member `SOIL::Element::insert` (`std::string uuid`, `std::shared_ptr< Element > child`)

Remove in future versions.

### Member `SOIL::Element::insert` (`std::string uuid`, `Element *child`)

Remove in future versions. This version takes a raw pointer (as e.g. returned by `new`) and internally makes a shared pointer.

### Member `SOIL::Enum::Enum` (`std::string value`)

Check whether this constructor is still meaningful in future releases. Usage is not recommended.

### Member `SOIL::Figure< T, x, y >::cast` (`T value`)

This function seems error prone in the multidimensional case.

### Class `SOIL::Object`

The `HTTP` handlers may be moved to protected if `HTTP::Server` is declared as friend class. Currently this is not done to allow for greater flexibility.

### Class `SOIL::Parameter< T, x, y >`

The `HTTP` handlers may be moved to protected if `HTTP::Server` is declared as friend class. Currently this is not done to allow for greater flexibility.

### Member `SOIL::Parameter< T, x, y >::mqtt` (`std::shared_ptr< MQTT::Publisher > publisher`, `int qos=0`, `bool retain=false`)

Currently the implementation of this function is redundant in `Variable` and `Parameter`. It is deliberately not moved to `Figure` as different implementations may occur in the future, but would be a valid option.

### Class `SOIL::Variable< T, x, y >`

The `HTTP` handlers may be moved to protected if `HTTP::Server` is declared as friend class. Currently this is not done to allow for greater flexibility.

### Member `SOIL::Variable< T, x, y >::hash`

This is still error prone across different languages and platforms as endianness and memory layout need to be considered.

### Member `SOIL::Variable< T, x, y >::mqtt` (`std::shared_ptr< MQTT::Publisher > publisher`, `int qos=0`, `bool retain=false`)

Currently the implementation of this function is redundant in `Variable` and `Parameter`. It is deliberately not moved to `Figure` as different implementations may occur in the future, but would be a valid option.



## Chapter 3

# Namespace Index

### 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">HTTP</a>	.....	<a href="#">15</a>
<a href="#">MQTT</a>	.....	<a href="#">16</a>
<a href="#">mqtt</a>	.....	<a href="#">17</a>
<a href="#">SIGN</a>	.....	<a href="#">17</a>
<a href="#">SOIL</a>	.....	
Type definitions	.....	<a href="#">17</a>
<a href="#">UDP</a>	.....	<a href="#">27</a>





## Chapter 4

# Hierarchical Index

### 4.1 Class Hierarchy

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

UDP::Broadcast . . . . .	29
MQTT::Configuration . . . . .	32
UDP::Configuration . . . . .	38
SOIL::Container< T, x, y > . . . . .	40
SOIL::Container< T, -1, -1 > . . . . .	46
SOIL::Container< T, x, -1 > . . . . .	48
SOIL::Container< T, x, y > . . . . .	40
SOIL::Enum . . . . .	62
SIGN::Hasher . . . . .	84
MQTT::MessageContainer . . . . .	88
MQTT::Publisher . . . . .	103
SOIL::Range< T > . . . . .	108
SOIL::Range< ENUM > . . . . .	111
SOIL::Range< std::string > . . . . .	112
HTTP::Resource . . . . .	114
SOIL::Element . . . . .	51
SOIL::Figure< T, -1, -1 > . . . . .	69
SOIL::Parameter< T, x, y > . . . . .	95
SOIL::Variable< T, x, y > . . . . .	132
SOIL::Figure< T, x, y > . . . . .	69
SOIL::Function . . . . .	76
SOIL::Object . . . . .	89
std::runtime_error	
MQTT::Exception . . . . .	65
UDP::Exception . . . . .	68
HTTP::Server . . . . .	122
SIGN::Signer . . . . .	124
SOIL::Time . . . . .	127



## Chapter 5

# Class Index

### 5.1 Class List

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

UDP::Broadcast	
UDP Broadcast	29
MQTT::Configuration	
MQTT publishing configuration	32
UDP::Configuration	
UDP Broadcast Configuration	38
SOIL::Container< T, x, y >	
Data Container	40
SOIL::Container< T, -1, -1 >	
Template specialization for Scalars	46
SOIL::Container< T, x, -1 >	
Template specialization for 1D-Arrays/Vectors	48
SOIL::Element	
SOIL Base Element	51
SOIL::Enum	
SOIL Enum Datatype	62
MQTT::Exception	
MQTT Publisher Exception	65
UDP::Exception	
UDP Broadcast Exception	68
SOIL::Figure< T, x, y >	
Intermediate class for <a href="#">Variable</a> and <a href="#">Parameter</a> that derives from <a href="#">Element</a>	69
SOIL::Function	
Function Class	76
SIGN::Hasher	
SHA256 Hasher	84
MQTT::MessageContainer	
Internal MQTT message struct	88
SOIL::Object	
Object Class	89
SOIL::Parameter< T, x, y >	
Parameter Class	95
MQTT::Publisher	
MQTT Publisher	103
SOIL::Range< T >	
Range Helper Class	108

---

SOIL::Range< ENUM >	
Enum Range	111
SOIL::Range< std::string >	
String Range	112
HTTP::Resource	
HTTP Resource base class	114
HTTP::Server	
HTPP Server	122
SIGN::Signer	
Signer	124
SOIL::Time	
SOIL Time	127
SOIL::Variable< T, x, y >	
Variable Class	132

## Chapter 6

# File Index

### 6.1 File List

Here is a list of all files with brief descriptions:

src/MQTT/Configuration.cpp	145
src/MQTT/Configuration.h	148
src/MQTT/constants.h	149
src/MQTT/LocalException.cpp	153
src/MQTT/LocalException.h	153
src/MQTT/MessageContainer.h	154
src/MQTT/Publisher.cpp	154
src/MQTT/Publisher.h	158
src/REST/constants.h	150
src/REST/Resource.cpp	159
src/REST/Resource.h	162
src/REST/Server.cpp	163
src/REST/Server.h	164
src/REST/Types.h	164
src/SIGN/constants.h	151
src/SIGN/Hasher.cpp	166
src/SIGN/Hasher.h	167
src/SIGN/Signer.cpp	168
src/SIGN/Signer.h	169
src/SOIL/constants.h	152
src/SOIL/Container.cpp	171
src/SOIL/Container.h	171
src/SOIL/Element.cpp	177
src/SOIL/Element.h	179
src/SOIL/Enum.cpp	181
src/SOIL/Enum.h	182
src/SOIL/Figure.cpp	183
src/SOIL/Figure.h	185
src/SOIL/Function.cpp	187
src/SOIL/Function.h	189
src/SOIL/json_helpers.cpp	191
src/SOIL/json_helpers.h	194
src/SOIL/Object.cpp	195
src/SOIL/Object.h	196
src/SOIL/Parameter.cpp	197

src/SOIL/Parameter.h	198
src/SOIL/Range.cpp	200
src/SOIL/Range.h	202
src/SOIL/Time.cpp	204
src/SOIL/Time.h	206
src/SOIL/Types.cpp	207
src/SOIL/Types.h	165
src/SOIL/Variable.cpp	207
src/SOIL/Variable.h	207
src/TEST/main.cpp	211
src/UDP/Broadcast.cpp	213
src/UDP/Broadcast.h	215
src/UDP/Configuration.cpp	147
src/UDP/Configuration.h	149
src/UDP/constants.h	152
src/UDP/Exception.cpp	216
src/UDP/Exception.h	216

## Chapter 7

# Namespace Documentation

### 7.1 HTTP Namespace Reference

#### Classes

- class [Resource](#)  
*HTTP Resource base class.*
- class [Server](#)  
*HTTP Server.*

#### Typedefs

- typedef web::http::http\_request [Request](#)  
*HTTP Request.*
- typedef web::http::http\_response [Response](#)  
*HTTP Response.*
- typedef web::http::status\_codes [Status](#)  
*HTTP Status.*
- typedef web::http::methods [Methods](#)  
*HTTP Methods.*
- typedef web::json::value [Json](#)  
*HTTP JSON.*

#### 7.1.1 Typedef Documentation

##### 7.1.1.1 Json

```
typedef web::json::value HTTP::Json
```

Typedefinition for Json documents as provided by the cpprestsdk library.

Definition at line [39](#) of file [Types.h](#).

### 7.1.1.2 Methods

```
typedef web::http::methods HTTP::Methods
```

Typedefinition for [HTTP Methods](#) (Enum) as provided by the cpprestsdk library.

Definition at line 32 of file [Types.h](#).

### 7.1.1.3 Request

```
typedef web::http::http_request HTTP::Request
```

Typedefinition for [HTTP Requests](#) as provided by the cpprestsdk library.

Definition at line 11 of file [Types.h](#).

### 7.1.1.4 Response

```
typedef web::http::http_response HTTP::Response
```

Typedefinition for [HTTP Responses](#) as provided by the cpprestsdk library.

Definition at line 18 of file [Types.h](#).

### 7.1.1.5 Status

```
typedef web::http::status_codes HTTP::Status
```

Typedefinition for [HTTP Status Codes](#) as provided by the cpprestsdk library.

Definition at line 25 of file [Types.h](#).

## 7.2 MQTT Namespace Reference

### Classes

- class [Configuration](#)  
*MQTT publishing configuration.*
- class [Exception](#)  
*MQTT Publisher Exception.*
- struct [MessageContainer](#)  
*Internal MQTT message struct.*
- class [Publisher](#)  
*MQTT Publisher.*



## 7.3 mqtt Namespace Reference

## 7.4 SIGN Namespace Reference

### Classes

- class [Hasher](#)  
*SHA256 Hasher.*
- class [Signer](#)  
*Signer.*

## 7.5 SOIL Namespace Reference

Type definitions.

### Classes

- class [Container](#)  
*Data Container.*
- class [Container< T, -1, -1 >](#)  
*Template specialization for Scalars.*
- class [Container< T, x, -1 >](#)  
*Template specialization for 1D-Arrays/Vectors.*
- class [Element](#)  
*SOIL Base Element.*
- class [Enum](#)  
*SOIL Enum Datatype.*
- class [Figure](#)  
*Intermediate class for [Variable](#) and [Parameter](#) that derives from [Element](#).*
- class [Function](#)  
*Function Class.*
- class [Object](#)  
*Object Class.*
- class [Parameter](#)  
*Parameter Class.*
- class [Range](#)  
*Range Helper Class.*
- class [Range< ENUM >](#)  
*Enum Range.*
- class [Range< std::string >](#)  
*String Range.*
- class [Time](#)  
*SOIL Time.*
- class [Variable](#)  
*Variable Class.*

## Typedefs

- typedef int64\_t [INT](#)  
*SOIL Integer.*
- typedef double [DOUBLE](#)  
*SOIL Double.*
- typedef bool [BOOL](#)  
*SOIL Boolean.*
- typedef std::string [STRING](#)  
*SOIL String.*
- typedef Enum [ENUM](#)  
*SOIL Enum.*
- typedef Time [TIME](#)  
*SOIL Time.*
- typedef std::vector< unsigned int > [DIMENSION](#)  
*SOIL Dimension.*

## Functions

- void [null\\_deleter](#) ([SOIL::Element](#) \*ptr)  
*Null deleter.*
- template<typename T >  
[HTTP::Json datatype](#) (void)  
*HTTP JSON datatype.*
- template<> DLL [HTTP::Json datatype< double >](#) (void)
- template<> DLL [HTTP::Json datatype< bool >](#) (void)
- template<> DLL [HTTP::Json datatype< std::string >](#) (void)
- template<> DLL [HTTP::Json datatype< int64\\_t >](#) (void)
- template<> DLL [HTTP::Json datatype< int >](#) (void)
- template<> DLL [HTTP::Json datatype< SOIL::TIME >](#) (void)
- template<> DLL [HTTP::Json datatype< SOIL::ENUM >](#) (void)
- template<typename T >  
DLL web::json::value [to\\_json](#) (const T &value)  
*Value to JSON.*
- template<typename T >  
DLL T [to\\_value](#) (web::json::value json)  
*JSON to Value.*
- template<> DLL web::json::value [to\\_json< double >](#) (const double &value)
- template<> DLL web::json::value [to\\_json< int64\\_t >](#) (const int64\_t &value)
- template<> DLL web::json::value [to\\_json< int >](#) (const int &value)
- template<> DLL web::json::value [to\\_json< std::string >](#) (const std::string &value)
- template<> DLL web::json::value [to\\_json< SOIL::TIME >](#) (const [SOIL::TIME](#) &value)
- template<> DLL web::json::value [to\\_json< SOIL::ENUM >](#) (const [SOIL::ENUM](#) &value)
- template<> DLL web::json::value [to\\_json< SOIL::BOOL >](#) (const bool &value)
- template<> DLL double [to\\_value< double >](#) (web::json::value value)
- template<> DLL int [to\\_value< int >](#) (web::json::value value)
- template<> DLL int64\_t [to\\_value< int64\\_t >](#) (web::json::value value)
- template<> DLL std::string [to\\_value< std::string >](#) (web::json::value json)
- template<> DLL [SOIL::TIME](#) [to\\_value< SOIL::TIME >](#) (web::json::value json)
- template<> DLL [SOIL::ENUM](#) [to\\_value< SOIL::ENUM >](#) (web::json::value json)
- template<> DLL bool [to\\_value< bool >](#) (web::json::value json)
- DLL bool [operator>=](#) (const [Time](#) &t1, const [Time](#) &t2)  
*GEQ Time Operator.*
- DLL bool [operator<=](#) (const [Time](#) &t1, const [Time](#) &t2)  
*LEQ Time Operator.*

### 7.5.1 Detailed Description

Set of type definitions to make sure the user can name [SOIL](#) specific types without knowing the explicit underlying C++ type.

### 7.5.2 Typedef Documentation

#### 7.5.2.1 BOOL

```
typedef bool SOIL::BOOL
```

Type definition for boolean type used in [SOIL](#).

Definition at line [33](#) of file [Types.h](#).

#### 7.5.2.2 DIMENSION

```
typedef std::vector<unsigned int> SOIL::DIMENSION
```

Type definition for the underlying datatype that is used to manage dimensions in [SOIL](#)

Definition at line [61](#) of file [Types.h](#).

#### 7.5.2.3 DOUBLE

```
typedef double SOIL::DOUBLE
```

Type definition for double type used in [SOIL](#). Double is the preferred floating point type in [SOIL](#).

Definition at line [26](#) of file [Types.h](#).

#### 7.5.2.4 ENUM

```
typedef Enum SOIL::ENUM
```

Type definition for the custom defined [Enum](#) in [SOIL](#) for consistency.

Definition at line [47](#) of file [Types.h](#).

#### 7.5.2.5 INT

```
typedef int64_t SOIL::INT
```

Type definition for integer type used in [SOIL](#).

Definition at line 18 of file [Types.h](#).

#### 7.5.2.6 STRING

```
typedef std::string SOIL::STRING
```

Type definition for string type used in [SOIL](#).

Definition at line 40 of file [Types.h](#).

#### 7.5.2.7 TIME

```
typedef Time SOIL::TIME
```

Type definition for the custom defined [Time](#) in [SOIL](#) for consistency.

Definition at line 54 of file [Types.h](#).

### 7.5.3 Function Documentation

#### 7.5.3.1 datatype()

```
template<typename T >
HTTP::Json SOIL::datatype (
    void )
```

Returns the datatype part of the SOIL-conformant JSON-representation of the figure, expressed as string. This function is declared in its specializations as it needs to be implemented explicitly for each datatype.

##### Template Parameters

<i>T</i>	Datatype under consideration
----------	------------------------------

##### Returns

[HTTP](#) JSON representaion as [HTTP::Json::string](#)

### 7.5.3.2 datatype< bool >()

```
template<>
HTTP::Json SOIL::datatype< bool > (
    void )
```

Definition at line 10 of file [Figure.cpp](#).

### 7.5.3.3 datatype< double >()

```
template<>
HTTP::Json SOIL::datatype< double > (
    void )
```

Definition at line 4 of file [Figure.cpp](#).

### 7.5.3.4 datatype< int >()

```
template<>
HTTP::Json SOIL::datatype< int > (
    void )
```

Definition at line 31 of file [Figure.cpp](#).

### 7.5.3.5 datatype< int64\_t >()

```
template<>
HTTP::Json SOIL::datatype< int64_t > (
    void )
```

Definition at line 24 of file [Figure.cpp](#).

### 7.5.3.6 datatype< SOIL::ENUM >()

```
template<>
DLL HTTP::Json SOIL::datatype< SOIL::ENUM > (
    void )
```

### 7.5.3.7 datatype< SOIL::TIME >()

```
template<>
DLL HTTP::Json SOIL::datatype< SOIL::TIME > (
    void )
```

### 7.5.3.8 datatype< std::string >()

```
template<>
DLL HTTP::Json SOIL::datatype< std::string > (
    void )
```

### 7.5.3.9 null\_deleter()

```
void SOIL::null_deleter (
    SOIL::Element * ptr ) [inline]
```

A function that essentially does nothing which is required as argument in the constructor of [Element](#) to avoid that delete is implicitly called a second time in the destructor when the member shared pointer to the object itself is deleted and the internal shared pointer is the last reference to the object. This situation occurs for root objects.

#### Parameters

in	<i>ptr</i>	Pointer on which to do nothing
----	------------	--------------------------------

Definition at line 278 of file [Element.h](#).

### 7.5.3.10 operator<=()

```
bool SOIL::operator<= (
    const Time & t1,
    const Time & t2 )
```

Friend Operator <=.

Lower or equal operation for two [Time](#) objects, which is needed for the correct implementation of the [Range](#) class.

Returns true if  $t1 \leq t2$ , otherwise false. If one of the [Time](#) objects is set to null, the operation returns false.

#### Parameters

<i>t1</i>	Left hand side time
<i>t2</i>	Right hand side time

**Returns**

Comparison result as boolean

Definition at line 128 of file [Time.cpp](#).

**7.5.3.11 operator>=()**

```
bool SOIL::operator>= (
    const Time & t1,
    const Time & t2 )
```

Friend Operator >=.

Greater or equal operation for two [Time](#) objects, which is needed for the correct implementation of the [Range](#) class.

Returns true if  $t1 \geq t2$ , otherwise false. If one of the [Time](#) objects is set to null, the operation returns false.

**Parameters**

<i>t1</i>	Left hand side time
<i>t2</i>	Right hand side time

**Returns**

Comparison result as boolean

Definition at line 116 of file [Time.cpp](#).

**7.5.3.12 to\_json()**

```
template<typename T >
DLL web::json::value SOIL::to_json (
    const T & value )
```

Helper function to convert a value to its correct JSON value (RHS of key in document). This is a template function that is specialized for every datatype.

**Template Parameters**

<i>T</i>	Datatype under consideration
----------	------------------------------

**Parameters**

in	<i>value</i>	Value to convert
----	--------------	------------------

**Returns**

JSON value

**7.5.3.13 to\_json< double >()**

```
template<>
web::json::value SOIL::to_json< double > (
    const double & value )
```

Definition at line 11 of file [json\\_helpers.cpp](#).

**7.5.3.14 to\_json< int >()**

```
template<>
web::json::value SOIL::to_json< int > (
    const int & value )
```

Definition at line 23 of file [json\\_helpers.cpp](#).

**7.5.3.15 to\_json< int64\_t >()**

```
template<>
web::json::value SOIL::to_json< int64_t > (
    const int64_t & value )
```

Definition at line 17 of file [json\\_helpers.cpp](#).

**7.5.3.16 to\_json< SOIL::BOOL >()**

```
template<>
DLL web::json::value SOIL::to_json< SOIL::BOOL > (
    const bool & value )
```

**7.5.3.17 to\_json< SOIL::ENUM >()**

```
template<>
DLL web::json::value SOIL::to_json< SOIL::ENUM > (
    const SOIL::ENUM & value )
```



**7.5.3.18 to\_json< SOIL::TIME >()**

```
template<>
DLL web::json::value SOIL::to_json< SOIL::TIME > (
    const SOIL::TIME & value )
```

**7.5.3.19 to\_json< std::string >()**

```
template<>
DLL web::json::value SOIL::to_json< std::string > (
    const std::string & value )
```

**7.5.3.20 to\_value()**

```
template<typename T >
DLL T SOIL::to_value (
    web::json::value json )
```

Helper function to convert a value from a JSON object to its equivalent internal [SOIL](#) datatype. This is a template function that is specialized for every datatype.

**Template Parameters**

<i>T</i>	Datatype under consideration
----------	------------------------------

**Parameters**

in	<i>json</i>	JSON object to convert
----	-------------	------------------------

**Returns**

Value in native datatype

**7.5.3.21 to\_value< bool >()**

```
template<>
bool SOIL::to_value< bool > (
    web::json::value json )
```

Definition at line 92 of file [json\\_helpers.cpp](#).

#### 7.5.3.22 to\_value< double >()

```
template<>
double SOIL::to_value< double > (
    web::json::value value )
```

Definition at line 55 of file [json\\_helpers.cpp](#).

#### 7.5.3.23 to\_value< int >()

```
template<>
int SOIL::to_value< int > (
    web::json::value value )
```

Definition at line 62 of file [json\\_helpers.cpp](#).

#### 7.5.3.24 to\_value< int64\_t >()

```
template<>
int64_t SOIL::to_value< int64_t > (
    web::json::value value )
```

Definition at line 68 of file [json\\_helpers.cpp](#).

#### 7.5.3.25 to\_value< SOIL::ENUM >()

```
template<>
DLL SOIL::ENUM SOIL::to_value< SOIL::ENUM > (
    web::json::value json )
```

#### 7.5.3.26 to\_value< SOIL::TIME >()

```
template<>
DLL SOIL::TIME SOIL::to_value< SOIL::TIME > (
    web::json::value json )
```

#### 7.5.3.27 to\_value< std::string >()

```
template<>
DLL std::string SOIL::to_value< std::string > (
    web::json::value json )
```

## 7.6 UDP Namespace Reference

### Classes

- class [Broadcast](#)  
*UDP Broadcast.*
- class [Configuration](#)  
*UDP Broadcast Configuration.*
- class [Exception](#)  
*UDP Broadcast Exception.*



## Chapter 8

# Class Documentation

### 8.1 UDP::Broadcast Class Reference

UDP Broadcast.

```
#include <Broadcast.h>
```

#### Public Member Functions

- [Broadcast](#) (int n\_threads=1)  
*Constructor.*
- [~Broadcast](#) ()  
*Destructor.*
- void [add\\_client](#) (std::string ip, unsigned int port)  
*Add UDP client.*
- void [remove\\_client](#) (std::string ip)  
*Remove UDP client.*
- void [send](#) (std::string message)  
*Send Message.*
- void [send](#) (std::vector< std::string > messages)  
*Send Multiple Message.*
- int [use\\_count](#) (void)  
*Number of Clients.*
- void [configure](#) (UDP::Configuration config)  
*Configure Broadcast.*

#### 8.1.1 Detailed Description

Simple [UDP Broadcast](#) implemented on top of boost asio. This is currently a convenience member of the C++ [SOIL](#) library as it's not part of the specification. It may be omitted in the future.

Definition at line 22 of file [Broadcast.h](#).

## 8.1.2 Constructor & Destructor Documentation

### 8.1.2.1 Broadcast()

```
UDP::Broadcast::Broadcast (
    int n_threads = 1 )
```

Constructor instantiating an [UDP Broadcast](#). The socket is immediately opened upon successful construction.

#### Parameters

in	<i>n_threads</i>	Number of threads to assign to the thread group. Only for very large loads a number of threads greater than 1 should be needed.
----	------------------	---

Definition at line 5 of file [Broadcast.cpp](#).

### 8.1.2.2 ~Broadcast()

```
UDP::Broadcast::~~Broadcast ( )
```

Destructor which closes the socket and stops all threads if not previously done.

Definition at line 23 of file [Broadcast.cpp](#).

## 8.1.3 Member Function Documentation

### 8.1.3.1 add\_client()

```
void UDP::Broadcast::add_client (
    std::string ip,
    unsigned int port )
```

Add a client to list of recipients of the [UDP](#) broadcast. One can only send to one port per IP address, the IPv4 address acts as unique identifier.

#### Parameters

in	<i>ip</i>	IPv4 address to send to
in	<i>port</i>	Port to send to

Definition at line 40 of file [Broadcast.cpp](#).

### 8.1.3.2 configure()

```
void UDP::Broadcast::configure (
    UDP::Configuration config )
```

Configure the broadcast with a configuration object, i.e. a list of endpoints.

#### Parameters

in	<i>config</i>	Configuration object
----	---------------	----------------------

Definition at line 132 of file [Broadcast.cpp](#).

### 8.1.3.3 remove\_client()

```
void UDP::Broadcast::remove_client (
    std::string ip )
```

Remove a client to list of recipients of the [UDP](#) broadcast which is identified through its IPv4 address.

#### Parameters

in	<i>ip</i>	IPv4 adress of the client to remove
----	-----------	-------------------------------------

Definition at line 63 of file [Broadcast.cpp](#).

### 8.1.3.4 send() [1/2]

```
void UDP::Broadcast::send (
    std::string message )
```

Send an individual message over the [UDP](#) Broadcast. It will be terminated by a newline (`\n`).

#### Parameters

in	<i>message</i>	Message to send
----	----------------	-----------------

Definition at line 109 of file [Broadcast.cpp](#).

### 8.1.3.5 send() [2/2]

```
void UDP::Broadcast::send (
    std::vector< std::string > messages )
```

Send a set of messages over the [UDP](#) Broadcast. They will be separated and terminated by a newlines (`\n`).

#### Parameters

in	<i>messages</i>	Messages to send
----	-----------------	------------------

Definition at line 114 of file [Broadcast.cpp](#).

### 8.1.3.6 use\_count()

```
int UDP::Broadcast::use_count (
    void ) [inline]
```

Get the number of registered [UDP](#) clients.

#### Returns

Number of registered [UDP](#) clients.

Definition at line 139 of file [Broadcast.h](#).

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

- [src/UDP/Broadcast.h](#)
- [src/UDP/Broadcast.cpp](#)

## 8.2 MQTT::Configuration Class Reference

[MQTT](#) publishing configuration.

```
#include <Configuration.h>
```

### Public Member Functions

- [Configuration](#) ()  
*Default constructor.*
- [Configuration](#) (std::string filename)  
*JSON Constructor.*
- [~Configuration](#) ()  
*Destructor.*
- std::string [uri](#) ()  
*URI Builder.*



## Public Attributes

- `std::string host`  
*Hostname of the [MQTT](#) broker.*
- `int port`  
*Port of the [MQTT](#) broker.*
- `std::string username`  
*Username for connecting to the [MQTT](#) broker.*
- `std::string password`  
*Password for connecting to the [MQTT](#) broker.*
- `bool clean_session`  
*Clean session flag.*
- `std::string root`  
*[MQTT](#) root topic.*
- `int keep_alive`  
*Keep alive interval in seconds.*
- `int min_delay_ms`  
*Minimum delay between to messages in milliseconds.*
- `int connection_timeout_s`  
*Connection timeout in seconds.*
- `bool ssl`  
*Use secured connection.*
- `bool verify`  
*Skip SSL verification.*
- `bool websocket`  
*Use websocket protocol.*
- `std::string path`  
*Websocket path.*
- `std::string certificate_authority`  
*Path to CA PEM-file.*

### 8.2.1 Detailed Description

Class acting as enhanced struct to accomodate all configuration options. Therefore all members are public to be directly accessible.

Definition at line 13 of file [Configuration.h](#).

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 Configuration() [1/2]

```
MQTT::Configuration::Configuration ( )
```

Default constructor applying the defaults documented.

Definition at line 8 of file [Configuration.cpp](#).

### 8.2.2.2 Configuration() [2/2]

```
MQTT::Configuration::Configuration (
    std::string filename )
```

Reads the configuration from a JSON-file. See the project assets for an example file.

#### Parameters

<i>in</i>	<i>filename</i>	Path to JSON-file (absolute or relative).
-----------	-----------------	---

Definition at line 27 of file [Configuration.cpp](#).

### 8.2.2.3 ~Configuration()

```
MQTT::Configuration::~~Configuration ( )
```

Standard Destructor

Definition at line 49 of file [Configuration.cpp](#).

## 8.2.3 Member Function Documentation

### 8.2.3.1 uri()

```
std::string MQTT::Configuration::uri ( )
```

Builds an URI for the underlying Paho-MQTT library.

#### Returns

Full URI string, e.g. `tcp://127.0.0.1:1883`.

Definition at line 53 of file [Configuration.cpp](#).

## 8.2.4 Member Data Documentation

#### 8.2.4.1 certificate\_authority

```
std::string MQTT::Configuration::certificate_authority
```

Path to a file containing the trusted certificate authorities in PEM-format for OpenSSL. This is mandatory when properly implementing secure [MQTT](#).

##### Precondition

Make sure that the hostname used for connecting is matching its name in the certificate chain.

Definition at line 133 of file [Configuration.h](#).

#### 8.2.4.2 clean\_session

```
bool MQTT::Configuration::clean_session
```

Flag whether to start a clean [MQTT](#) session. Defaults to `true`.

Definition at line 55 of file [Configuration.h](#).

#### 8.2.4.3 connection\_timeout\_s

```
int MQTT::Configuration::connection_timeout_s
```

Number of seconds to wait until the connection succeeds. This influence the length of the blocking call during connection. Defaults to 30.

Definition at line 90 of file [Configuration.h](#).

#### 8.2.4.4 host

```
std::string MQTT::Configuration::host
```

Adress or hostname of the [MQTT](#) broker. Can be an FQDN or IP address. Defaults to `127.0.0.1`.

Definition at line 22 of file [Configuration.h](#).

#### 8.2.4.5 keep\_alive

```
int MQTT::Configuration::keep_alive
```

Keep alive interval in seconds for the [MQTT](#) client. Defaults to 30 .

Definition at line 72 of file [Configuration.h](#).

#### 8.2.4.6 min\_delay\_ms

```
int MQTT::Configuration::min_delay_ms
```

A minimum delay which should be kept between sending two messages, provided in milliseconds. Defaults to 0 .

##### Postcondition

This is not enforced in the client, but should be called as property of the publishing by the using code.

Definition at line 81 of file [Configuration.h](#).

#### 8.2.4.7 password

```
std::string MQTT::Configuration::password
```

Password when connecting to the [MQTT](#) broker. Be mindful with passwords when committing to version control! Defaults to `guest` .

Definition at line 47 of file [Configuration.h](#).

#### 8.2.4.8 path

```
std::string MQTT::Configuration::path
```

Websocket path that forms the last part of the URI, e.g. `/mqtt-ws/`. Defaults to `" "` .

Definition at line 124 of file [Configuration.h](#).

#### 8.2.4.9 port

```
int MQTT::Configuration::port
```

Port to use for the [MQTT](#) broker. Defaults to 1883 in the constructor. Typical ports are 1883 ([MQTT](#)), 8883 ([MQTT](#) over TLS) and 443 ([MQTT](#) over secure websockets).

Definition at line 30 of file [Configuration.h](#).

#### 8.2.4.10 root

```
std::string MQTT::Configuration::root
```

Root topic that is prepended to any published topic. This is intended for cases where permissions enforce that you publish under a defined root topic. The default is no root topic prefix.

Definition at line 64 of file [Configuration.h](#).

#### 8.2.4.11 ssl

```
bool MQTT::Configuration::ssl
```

Boolean flag whether to use a secured connection ([MQTT](#) over TLS or secure websocket). Defaults to `false`.

##### Precondition

If not ignoring verification, trusted root certificated must be provided in a PEM-file.

Definition at line 99 of file [Configuration.h](#).

#### 8.2.4.12 username

```
std::string MQTT::Configuration::username
```

Username when connecting to the [MQTT](#) broker. When using RabbitMQ, make sure to consider the VHOST prefix. Defaults to `guest`.

Definition at line 39 of file [Configuration.h](#).

#### 8.2.4.13 verify

```
bool MQTT::Configuration::verify
```

Boolean flag whether to ignore SSL certificate validation, i.e. for hostname matching and trusted authority. This should only be used for basic development purposes.

Definition at line 107 of file [Configuration.h](#).

#### 8.2.4.14 websocket

```
bool MQTT::Configuration::websocket
```

Boolean flag whether to use the websocket extension of [MQTT](#). Make sure to specify the correct path. Defaults to `false`.

Definition at line 116 of file [Configuration.h](#).

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

- [src/MQTT/Configuration.h](#)
- [src/MQTT/Configuration.cpp](#)

## 8.3 UDP::Configuration Class Reference

[UDP Broadcast Configuration](#).

```
#include <Configuration.h>
```

### Public Types

- typedef std::map< std::string, int >::iterator [iterator](#)  
*Iterator.*

### Public Member Functions

- [Configuration](#) ()  
*Constructor.*
- [Configuration](#) (std::string filename)  
*JSON Constructor.*
- [~Configuration](#) ()  
*Destructor.*

### Public Attributes

- std::map< std::string, int > [clients](#)  
*List of clients.*

#### 8.3.1 Detailed Description

Class to simplify the configuration of [UDP](#) broadcasts, i.e. by providing a list of client endpoints that can be populated from a JSON file. For simplicity, all members are public.

Definition at line 13 of file [Configuration.h](#).

## 8.3.2 Member Typedef Documentation

### 8.3.2.1 iterator

```
typedef std::map<std::string,int>::iterator UDP::Configuration::iterator
```

Typedefinition to provide an iterator-sytlled access

Definition at line 21 of file [Configuration.h](#).

## 8.3.3 Constructor & Destructor Documentation

### 8.3.3.1 Configuration() [1/2]

```
UDP::Configuration::Configuration ( )
```

Default constructor, creates an empty list of clients.

Definition at line 7 of file [Configuration.cpp](#).

### 8.3.3.2 Configuration() [2/2]

```
UDP::Configuration::Configuration (
    std::string filename )
```

Constructor the takes a JSON file as input to populate the list of clients. See the project assets for a sample file.

#### Parameters

in	<i>filename</i>	Path to the JSON configuration file
----	-----------------	-------------------------------------

Definition at line 12 of file [Configuration.cpp](#).

### 8.3.3.3 ~Configuration()

```
UDP::Configuration::~~Configuration ( )
```

Default destructor, empties the map.

Definition at line 24 of file [Configuration.cpp](#).

### 8.3.4 Member Data Documentation

#### 8.3.4.1 clients

```
std::map<std::string, int> UDP::Configuration::clients
```

List of clients, implemented as map where the IPv4 address acts as unique identifier.

Definition at line 28 of file [Configuration.h](#).

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

- [src/UDP/Configuration.h](#)
- [src/UDP/Configuration.cpp](#)

## 8.4 SOIL::Container< T, x, y > Class Template Reference

Data [Container](#).

```
#include <Container.h>
```

### Public Member Functions

- [Container](#) ()  
*Empty constructor.*
- [Container](#) (const std::vector< std::vector< T > > &value)  
*Data copy constructor.*
- [Container](#) (HTTP::Json json)  
*JSON Constructor.*
- std::vector< std::vector< T > > [operator\\*](#) (void) const  
*Deferencing operator.*
- bool [is\\_null](#) (void) const  
*Is Null?*
- void [set\\_null](#) (bool \_null=true)  
*Set null.*
- [HTTP::Json wjson](#) (void)  
*WJSON representation.*
- bool [check\\_range](#) ([Range](#)< T > range) const  
*Check range.*
- T & [at](#) (int i, int j)  
*Data Accessor.*
- std::vector< unsigned char > [serialize\\_value](#) (void) const  
*Serialize value.*
- std::vector< unsigned char > [serialize\\_dimensions](#) (void) const  
*Serialize dimensions.*



### 8.4.1 Detailed Description

```
template<typename T, int x = -1, int y = -1>  
class SOIL::Container< T, x, y >
```

[Container](#) class to manage multidimensional data of (nearly) arbitrary type. The data can be scalar, 1D or 2D while the size of dimensions is a priori unknown. This class then provides an abstraction layer to the other elements of [SOIL](#) to allow for consistent implementation. Therefor it makes heavy use of templates, which are specialized for certain cases where  $x$  and/or  $y$  are -1 and for special types.

## Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

Definition at line 21 of file [Container.h](#).

## 8.4.2 Constructor & Destructor Documentation

### 8.4.2.1 Container() [1/3]

```
template<typename T >
SOIL::Container< T >::Container
```

Constructs an empty container. `_null` will be set to true as no data has been added.

## Exceptions

<code>std::runtime_error</code>	An exception is thrown if an invalid combination of template parameters is used.
---------------------------------	--

Definition at line 196 of file [Container.h](#).

### 8.4.2.2 Container() [2/3]

```
template<typename T , int x, int y>
SOIL::Container< T, x, y >::Container (
    const std::vector< std::vector< T > > & value )
```

Constructor that initializes the data with the given value.

## Parameters

<i>in</i>	<i>value</i>	Data for initialization, wich must match in type and dimension
-----------	--------------	--

Definition at line 210 of file [Container.h](#).

### 8.4.2.3 Container() [3/3]

```
template<typename T >
SOIL::Container< T >::Container (
    HTTP::Json json )
```

Constructor that takes dimension and value from an JSON representation adhering to the [SOIL](#) nomenclature.

#### Parameters

in	<i>json</i>	JSON object to parse
----	-------------	----------------------

Definition at line 231 of file [Container.h](#).

### 8.4.3 Member Function Documentation

#### 8.4.3.1 at()

```
template<typename T , int x, int y>
T & SOIL::Container< T >::at (
    int i,
    int j )
```

STL-style data accesor retrieving a a single element from multidimensional-data.

#### Parameters

in	<i>i</i>	Index position along first dimension
in	<i>j</i>	Index position along second dimension

#### Returns

Data value of type T

Definition at line 313 of file [Container.h](#).

#### 8.4.3.2 check\_range()

```
template<typename T >
bool SOIL::Container< T >::check_range (
    Range< T > range ) const
```

Check whether the data of the container lies within the specified range. In the case of multidimensional data, this check is carried out on all elememts and only returns true if it is applicable to all.

#### Parameters

in	<i>range</i>	<a href="#">Range</a> for which to check
----	--------------	--

**Returns**

Check result as boolean

Definition at line 293 of file [Container.h](#).

**8.4.3.3 is\_null()**

```
template<typename T , int x = -1, int y = -1>
bool SOIL::Container< T, x, y >::is_null (
    void ) const [inline]
```

**Function** that returns true if the current data is set to null and false else.

**Returns**

Null flag

Definition at line 79 of file [Container.h](#).

**8.4.3.4 operator\*()**

```
template<typename T >
T SOIL::Container< T >::operator* (
    void ) const
```

Return a copy of the raw data, similar to other uses of \* in the standard library.

**Returns**

Copy of underlying data.

Definition at line 263 of file [Container.h](#).

**8.4.3.5 serialize\_dimensions()**

```
template<typename T >
std::vector< unsigned char > SOIL::Container< T >::serialize_dimensions (
    void ) const
```

Serialize the dimensions to a bytestring for hashing purposes

**Returns**

Bytestring of serialization

Definition at line 350 of file [Container.h](#).

#### 8.4.3.6 serialize\_value()

```
template<typename T >
std::vector< unsigned char > SOIL::Container< T >::serialize_value (
    void ) const
```

Serialize the value to a bytestring for hashing purposes

##### Returns

Bytestring of serialization

Definition at line 323 of file [Container.h](#).

#### 8.4.3.7 set\_null()

```
template<typename T , int x = -1, int y = -1>
void SOIL::Container< T, x, y >::set_null (
    bool _null = true ) [inline]
```

[Function](#) to set the null status of the data container

##### Parameters

in	<a href="#">_null</a>	Boolean state flag
----	-----------------------	--------------------

Definition at line 87 of file [Container.h](#).

#### 8.4.3.8 wjson()

```
template<typename T >
HTTP::Json SOIL::Container< T >::wjson (
    void )
```

Get a JSON object representation of the container's current data. The naming [wjson\(\)](#) is chosen as it is consistently used through the library when using wide-string JSON as introduced by [cpprestsdk](#).

##### Returns

JSON object.

Definition at line 269 of file [Container.h](#).

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

- [src/SOIL/Container.h](#)

## 8.5 SOIL::Container< T, -1, -1 > Class Template Reference

Template specialization for Scalars.

```
#include <Container.h>
```

### Public Member Functions

- [Container](#) ()
- [Container](#) (const T &value)
- [Container](#) (HTTP::Json json)
- T [operator\\*](#) (void) const
- bool [is\\_null](#) (void) const
- void [set\\_null](#) (bool \_null=true)
- [HTTP::Json wjson](#) (void)
- bool [check\\_range](#) ([Range](#)< T > range) const
- T & [at](#) (void)
- std::vector< unsigned char > [serialize\\_value](#) (void) const
- std::vector< unsigned char > [serialize\\_dimensions](#) (void) const

### 8.5.1 Detailed Description

```
template<typename T>
class SOIL::Container< T, -1, -1 >
```

This is the specialization of the templated class for scalar data. For the documentation of methods, please refer to the unpecialized class.

#### Template Parameters

<i>T</i>	Type of the data
----------	------------------

Definition at line 174 of file [Container.h](#).

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 Container() [1/3]

```
template<typename T >
SOIL::Container< T, -1, -1 >::Container ( )
```

### 8.5.2.2 Container() [2/3]

```
template<typename T >
SOIL::Container< T, -1, -1 >::Container (
    const T & value )
```

### 8.5.2.3 Container() [3/3]

```
template<typename T >
SOIL::Container< T, -1, -1 >::Container (
    HTTP::Json json )
```

## 8.5.3 Member Function Documentation

### 8.5.3.1 at()

```
template<typename T >
T & SOIL::Container< T, -1, -1 >::at (
    void )
```

### 8.5.3.2 check\_range()

```
template<typename T >
bool SOIL::Container< T, -1, -1 >::check_range (
    Range< T > range ) const
```

### 8.5.3.3 is\_null()

```
template<typename T >
bool SOIL::Container< T, -1, -1 >::is_null (
    void ) const [inline]
```

Definition at line 184 of file [Container.h](#).

#### 8.5.3.4 operator\*()

```
template<typename T >
T SOIL::Container< T, -1, -1 >::operator* (
    void ) const
```

#### 8.5.3.5 serialize\_dimensions()

```
template<typename T >
std::vector< unsigned char > SOIL::Container< T, -1, -1 >::serialize_dimensions (
    void ) const
```

#### 8.5.3.6 serialize\_value()

```
template<typename T >
std::vector< unsigned char > SOIL::Container< T, -1, -1 >::serialize_value (
    void ) const
```

#### 8.5.3.7 set\_null()

```
template<typename T >
void SOIL::Container< T, -1, -1 >::set_null (
    bool _null = true ) [inline]
```

Definition at line 185 of file [Container.h](#).

#### 8.5.3.8 wjson()

```
template<typename T >
HTTP::Json SOIL::Container< T, -1, -1 >::wjson (
    void )
```

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

- [src/SOIL/Container.h](#)

## 8.6 SOIL::Container< T, x, -1 > Class Template Reference

Template specialization for 1D-Arrays/Vectors.

```
#include <Container.h>
```



## Public Member Functions

- [Container](#) ()
- [Container](#) (const std::vector< T > &value)
- [Container](#) (HTTP::Json json)
- std::vector< T > [operator\\*](#) (void) const
- bool [is\\_null](#) (void) const
- void [set\\_null](#) (bool \_null=true)
- [HTTP::Json wjson](#) (void)
- bool [check\\_range](#) ([Range](#)< T > range) const
- T & [at](#) (int i)
- std::vector< unsigned char > [serialize\\_value](#) (void) const
- std::vector< unsigned char > [serialize\\_dimensions](#) (void) const

### 8.6.1 Detailed Description

```
template<typename T, int x>
class SOIL::Container< T, x, -1 >
```

This is the specialization of the templated class for data that expands along one dimension. For the documentation of methods, please refer to the unpecialized class.

#### Template Parameters

<i>T</i>	Type of the data
<i>x</i>	First dimension of the data

Definition at line 146 of file [Container.h](#).

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 Container() [1/3]

```
template<typename T , int x>
SOIL::Container< T, x, -1 >::Container ( )
```

#### 8.6.2.2 Container() [2/3]

```
template<typename T , int x>
SOIL::Container< T, x, -1 >::Container (
    const std::vector< T > & value )
```

### 8.6.2.3 Container() [3/3]

```
template<typename T , int x>
SOIL::Container< T, x, -1 >::Container (
    HTTP::Json json )
```

## 8.6.3 Member Function Documentation

### 8.6.3.1 at()

```
template<typename T , int x>
T & SOIL::Container< T, x, -1 >::at (
    int i )
```

### 8.6.3.2 check\_range()

```
template<typename T , int x>
bool SOIL::Container< T, x, -1 >::check_range (
    Range< T > range ) const
```

### 8.6.3.3 is\_null()

```
template<typename T , int x>
bool SOIL::Container< T, x, -1 >::is_null (
    void ) const [inline]
```

Definition at line 156 of file [Container.h](#).

### 8.6.3.4 operator\*()

```
template<typename T , int x>
std::vector< T > SOIL::Container< T, x, -1 >::operator* (
    void ) const
```

### 8.6.3.5 serialize\_dimensions()

```
template<typename T , int x>
std::vector< unsigned char > SOIL::Container< T, x, -1 >::serialize_dimensions (
    void ) const
```

### 8.6.3.6 serialize\_value()

```
template<typename T , int x>
std::vector< unsigned char > SOIL::Container< T, x, -1 >::serialize_value (
    void ) const
```

### 8.6.3.7 set\_null()

```
template<typename T , int x>
void SOIL::Container< T, x, -1 >::set_null (
    bool _null = true ) [inline]
```

Definition at line 157 of file [Container.h](#).

### 8.6.3.8 wjson()

```
template<typename T , int x>
HTTP::Json SOIL::Container< T, x, -1 >::wjson (
    void )
```

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

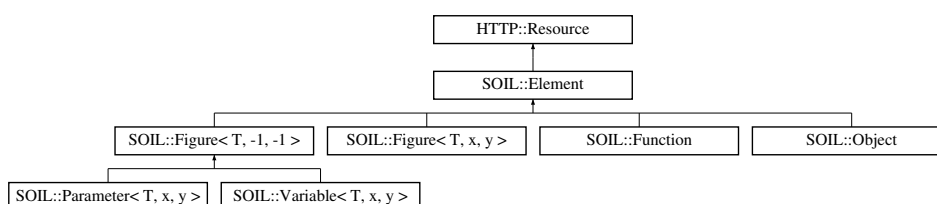
- [src/SOIL/Container.h](#)

## 8.7 SOIL::Element Class Reference

[SOIL](#) Base [Element](#).

```
#include <Element.h>
```

Inheritance diagram for SOIL::Element:



## Public Member Functions

- [Element](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [ontology](#)="")  
*Constructor.*
- virtual [~Element](#) ()  
*Destructor.*
- std::vector< std::string > [fqid](#) (void)  
*FQID.*
- std::shared\_ptr< [Element](#) > [operator\[\]](#) (std::string [fqid](#))  
*Access Operator.*
- std::shared\_ptr< [Element](#) > [add](#) (std::string [uuid](#), std::shared\_ptr< [Element](#) > [child](#))  
*Add Child [Element](#).*
- std::shared\_ptr< [Element](#) > [add](#) (std::string [uuid](#), [Element](#) \*[child](#))  
*Add Child [Element](#).*
- bool [insert](#) (std::string [uuid](#), std::shared\_ptr< [Element](#) > [child](#))  
*Add Child [Element](#).*
- bool [insert](#) (std::string [uuid](#), [Element](#) \*[child](#))  
*Add Child [Element](#).*
- bool [remove](#) (std::string [uuid](#))  
*Remove Child element.*
- template<typename T >  
T \* [cast](#) (void)  
*Get dynamically casted pointer.*
- bool [is\\_object](#) (void) const  
*Is [Object](#)?*
- bool [is\\_variable](#) (void) const  
*Is [Variable](#)?*
- bool [is\\_function](#) (void) const  
*Is [Function](#)?*
- bool [is\\_parameter](#) (void) const  
*Is [Parameter](#)?*
- virtual [HTTP::Json wjson](#) (void)  
*[HTTP](#) JSON.*
- virtual std::string [json](#) (void)  
*JSON string.*
- [HTTP::Response handle](#) ([HTTP::Request](#) [request](#), std::smatch [match](#)=std::smatch())  
*[HTTP](#) Handler.*

## Public Attributes

- std::map< std::string, std::shared\_ptr< [Element](#) > > [children](#)  
*Children Map.*
- std::shared\_ptr< [Element](#) > [parent](#)  
*Parent Pointer.*
- std::shared\_ptr< [Element](#) > [self](#)  
*Self Pointer.*
- std::string [uuid](#)  
*Local UUID.*
- std::string [name](#)

- Name.*
- std::string [description](#)
- Description.*
- std::string [ontology](#)
- Ontology identifier.*
- std::recursive\_mutex [mutex](#)
- Element Mutex.*

## Additional Inherited Members

### 8.7.1 Detailed Description

[Element](#) is the base class of [Object](#), [Function](#), [Parameter](#) and [Variable](#) in [SOIL](#). It is the first class that inherits from [HTTP::Resource](#). It contains the main implementation of the business logic for building the [Element](#) tree, i.e. managing child items and resolving absolute and relative parths. It also provides common base class which can be used for generic pointer in C++.

Definition at line 23 of file [Element.h](#).

### 8.7.2 Constructor & Destructor Documentation

#### 8.7.2.1 Element()

```
SOIL::Element::Element (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string ontology = "" )
```

Default constructor for [SOIL](#) elements, to be called from derived classes.

##### Parameters

in	<i>parent</i>	Shared pointer to parent object. Can be set to NULL for the creation of a root object
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed

Definition at line 5 of file [Element.cpp](#).

#### 8.7.2.2 ~Element()

```
SOIL::Element::~~Element ( ) [virtual]
```

Destructor for [Element](#), which is declared virtual such that [Element](#) pointers can be used to delete instances of derived types. When an [Element](#) is deleted in [SOIL](#), all references to the child objects are cleared.

#### Postcondition

If the parent element was the last instance to contain shared pointers to the child elements, their lifecycle will end with this operation.

Definition at line 19 of file [Element.cpp](#).

## 8.7.3 Member Function Documentation

### 8.7.3.1 `add()` [1/2]

```
std::shared_ptr< SOIL::Element > SOIL::Element::add (
    std::string uuid,
    Element * child )
```

Inserts a child to this element. Children are always managed as shared pointers, i.e. no object is copied. This version takes a raw pointer (as e.g. returned by `new`) and internally makes a shared pointer.

#### Parameters

in	<i>uuid</i>	UUID to assign to the child, which also acts as internal identifier
in	<i>child</i>	Pointer to the element

#### Returns

Shared pointer to child element

Definition at line 86 of file [Element.cpp](#).

### 8.7.3.2 `add()` [2/2]

```
std::shared_ptr< SOIL::Element > SOIL::Element::add (
    std::string uuid,
    std::shared_ptr< Element > child )
```

Inserts a child to this element. Children are always managed as shared pointers, i.e. no object is copied.

#### Parameters

in	<i>uuid</i>	UUID to assign to the child, which also acts as internal identifier
in	<i>child</i>	Shared pointer to the element

**Returns**

Shared pointer to child element

Definition at line 79 of file [Element.cpp](#).

**8.7.3.3 cast()**

```
template<typename T >
T * SOIL::Element::cast (
    void )
```

Dynamically casts the stored element reference to a given type. This useful for casting pointers to derived classes of which the type is known.

**Template Parameters**

<i>T</i>	Derived type to cast to.
----------	--------------------------

**Returns**

Raw pointer

Definition at line 264 of file [Element.h](#).

**8.7.3.4 fqid()**

```
std::vector< std::string > SOIL::Element::fqid (
    void )
```

Constructs the FQID of the element by concatenating the UUIDs of the parent elements

**Returns**

FQID string separated by /

Definition at line 26 of file [Element.cpp](#).

**8.7.3.5 handle()**

```
HTTP::Response SOIL::Element::handle (
    HTTP::Request request,
    std::smatch match = std::smatch() ) [virtual]
```

Handles an incoming [HTTP](#) request as resource. It determines the correct child item (or the item itself) and then calls the handler of the [HTTP::Resource](#) base class. The arguments are directly passed on to the latter.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by <code>cpprestsdk</code>
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by `cpprestsdk`

Reimplemented from [HTTP::Resource](#).

Definition at line 155 of file [Element.cpp](#).

**8.7.3.6 insert() [1/2]**

```
bool SOIL::Element::insert (
    std::string uuid,
    Element * child )
```

Alias to add provide compatibilty to prior versions.

**Todo** Remove in future versions. This version takes a raw pointer (as e.g. returned by `new`) and internally makes a shared pointer.

**Parameters**

in	<i>uuid</i>	UUID to assign to the child, which also acts as internal identifier
in	<i>child</i>	Pointer to the element

**Returns**

True if the element previously already existed.

Definition at line 98 of file [Element.cpp](#).

**8.7.3.7 insert() [2/2]**

```
bool SOIL::Element::insert (
    std::string uuid,
    std::shared_ptr< Element > child )
```

Alias to add provide compatibilty to prior versions.

**Todo** Remove in future versions.



**Parameters**

in	<i>uuid</i>	UUID to assign to the child, which also acts as internal identifier
in	<i>child</i>	Shared pointer to the element

**Returns**

True if the element previously already existed.

Definition at line 91 of file [Element.cpp](#).

**8.7.3.8 is\_function()**

```
bool SOIL::Element::is_function (
    void ) const
```

Determines whether the element actually is a function. Internally, the prefix FUN- of the UUID is checked for this purpose. This function is typically used before dynamic casts.

**Returns**

Boolean response whether the element is a function.

Definition at line 120 of file [Element.cpp](#).

**8.7.3.9 is\_object()**

```
bool SOIL::Element::is_object (
    void ) const
```

Determines whether the element actually is an object. Internally, the prefix OBJ- of the UUID is checked for this purpose. This function is typically used before dynamic casts.

**Returns**

Boolean response whether the element is an object.

Definition at line 110 of file [Element.cpp](#).

#### 8.7.3.10 is\_parameter()

```
bool SOIL::Element::is_parameter (
    void ) const
```

Determines whether the element actually is a parameter. Internally, the prefix PAR- of the UUID is checked for this purpose. This function is typically used before dynamic casts.

##### Returns

Boolean response whether the element is a parameter.

Definition at line 125 of file [Element.cpp](#).

#### 8.7.3.11 is\_variable()

```
bool SOIL::Element::is_variable (
    void ) const
```

Determines whether the element actually is a variable. Internally, the prefix VAR- of the UUID is checked for this purpose. This function is typically used before dynamic casts.

##### Returns

Boolean response whether the element is at.

Definition at line 115 of file [Element.cpp](#).

#### 8.7.3.12 json()

```
std::string SOIL::Element::json (
    void ) [virtual]
```

Get a JSON string corresponding to the current state of the element. This function internally useses [wjson\(\)](#) such that there is no need to override both methods in derived classes.

##### Returns

JSON string

Definition at line 148 of file [Element.cpp](#).

#### 8.7.3.13 operator[]()

```
std::shared_ptr< SOIL::Element > SOIL::Element::operator[] (
    std::string fqid )
```

Returns the child matching the relative FQID of the element. If the empty string is passed, the element itself is returned.

**Exceptions**

<code>std::runtime_error</code>	If no matching child is found, a <code>runtime_error</code> is thrown.
---------------------------------	--

**Parameters**

<code>in</code>	<code>fqid</code>	Relative FQID of the element to retrieve
-----------------	-------------------	--

**Returns**

Share pointer to the requested element

Definition at line 38 of file [Element.cpp](#).

**8.7.3.14 remove()**

```
bool SOIL::Element::remove (
    std::string uuid )
```

Removes an element from the children of this element. If the internal shared pointer is the last reference to this element, it will go out of scope.

**Parameters**

<code>in</code>	<code>uuid</code>	UUID to identify the element to remove
-----------------	-------------------	--

**Returns**

True if the element existed prior to deletion.

Definition at line 103 of file [Element.cpp](#).

**8.7.3.15 wjson()**

```
HTTP::Json SOIL::Element::wjson (
    void ) [virtual]
```

Get a [HTTP](#) JSON object corresponding to the current state of the element. This function is virtual as normally it should be specialized by [Object](#), [Variable](#), [Parameter](#) and [Function](#).

**Returns**

JSON object

Reimplemented in [SOIL::Function](#), [SOIL::Object](#), [SOIL::Figure< T, x, y >](#), [SOIL::Figure< T, -1, -1 >](#), [SOIL::Parameter< T, x, y >](#), and [SOIL::Variable< T, x, y >](#).

Definition at line 130 of file [Element.cpp](#).

## 8.7.4 Member Data Documentation

### 8.7.4.1 children

```
std::map<std::string, std::shared_ptr<Element> > SOIL::Element::children
```

Map of child elements, which are referenced by means of shared pointers and identified by their local UUID in string form.

Definition at line 32 of file [Element.h](#).

### 8.7.4.2 description

```
std::string SOIL::Element::description
```

Human-readable description of the element in string format.

Definition at line 67 of file [Element.h](#).

### 8.7.4.3 mutex

```
std::recursive_mutex SOIL::Element::mutex
```

Recursive Mutex to provide thread safe access to [SOIL](#) elements.

Definition at line 83 of file [Element.h](#).

### 8.7.4.4 name

```
std::string SOIL::Element::name
```

Human-readable name of the element in string format.

Definition at line 60 of file [Element.h](#).

#### 8.7.4.5 ontology

```
std::string SOIL::Element::ontology
```

Ontology identifier, can be set to null if no ontology is followed

##### Precondition

Referencing to an ontology prerequisites that an ontology in [SOIL](#) format has been declared elsewhere.

Definition at line 76 of file [Element.h](#).

#### 8.7.4.6 parent

```
std::shared_ptr<Element> SOIL::Element::parent
```

Shared pointer reference to the parent element.

Definition at line 39 of file [Element.h](#).

#### 8.7.4.7 self

```
std::shared_ptr<Element> SOIL::Element::self
```

Shared pointer reference to the element itself.

Definition at line 46 of file [Element.h](#).

#### 8.7.4.8 uuid

```
std::string SOIL::Element::uuid
```

Locally unique identifier of the object, which must start with OBJ-, FUN-, PAR-, or VAR- depending on the type.

Definition at line 53 of file [Element.h](#).

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

- [src/SOIL/Element.h](#)
- [src/SOIL/Element.cpp](#)

## 8.8 SOIL::Enum Class Reference

SOIL Enum Datatype.

```
#include <Enum.h>
```

### Public Member Functions

- [Enum](#) ()  
*Constructor.*
- [Enum](#) (std::string value)  
*Constructor initialized with value.*
- [Enum](#) (std::string value, std::vector< std::string > [choices](#))  
*Initializing Container.*
- [~Enum](#) ()  
*Destructor.*
- std::string [selected](#) (void) const  
*Get selected value.*
- std::vector< std::string > [choices](#) (void)  
*Get Choices.*
- int [index](#) () const  
*Get index of selected element.*
- int [index](#) (std::string value) const  
*Determine index of value.*
- void [set](#) (int value)  
*Set selected item (index)*
- void [set](#) (std::string value)  
*Set selected item (value)*

### 8.8.1 Detailed Description

C++ class to represent the [SOIL Enum](#) datatype. Internally it is based on strings.

Definition at line 15 of file [Enum.h](#).

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 Enum() [1/3]

```
SOIL::Enum::Enum ( )
```

Standard constructor which leaves everything uninitialized.

Definition at line 4 of file [Enum.cpp](#).

#### 8.8.2.2 Enum() [2/3]

```
SOIL::Enum::Enum (
    std::string value )
```

Constructor that initializes the enumeration with a selected value.

**Todo** Check whether this constructor is still meaningful in future releases. Usage is not recommended.

## Parameters

in	<i>value</i>	Value for initialization
----	--------------	--------------------------

Definition at line 8 of file [Enum.cpp](#).

**8.8.2.3 Enum()** [3/3]

```
SOIL::Enum::Enum (
    std::string value,
    std::vector< std::string > choices )
```

Constructor that initializes the enumeration with choices and a value.

## Parameters

in	<i>value</i>	Value for initialization
in	<i>choices</i>	Applicable set of choices for the enumeration

Definition at line 12 of file [Enum.cpp](#).

**8.8.2.4 ~Enum()**

```
SOIL::Enum::~Enum ( )
```

Standard destructor without special efforts.

Definition at line 17 of file [Enum.cpp](#).

**8.8.3 Member Function Documentation****8.8.3.1 choices()**

```
std::vector< std::string > SOIL::Enum::choices (
    void )
```

Get the choices that are available for this enumeration.

## Returns

Available choices as vector of strings

Definition at line 26 of file [Enum.cpp](#).

### 8.8.3.2 index() [1/2]

```
int SOIL::Enum::index ( ) const
```

Traversed the vector of choices and return the index of the current element.

#### Returns

Index of the selected element

Definition at line 31 of file [Enum.cpp](#).

### 8.8.3.3 index() [2/2]

```
int SOIL::Enum::index (
    std::string value ) const
```

Converts a string value to its integer value. This function is useful when interacting with C++ ENUMs consisting of integers.

#### Exceptions

<code>std::logic_error</code>	If the value is not found, an exception is thrown.
-------------------------------	--

#### Parameters

in	value	Value to search
----	-------	-----------------

#### Returns

Index of the searched value

Definition at line 43 of file [Enum.cpp](#).

### 8.8.3.4 selected()

```
std::string SOIL::Enum::selected (
    void ) const
```

Get the value that the enum currently holds as selected.

#### Returns

Current value

Definition at line 21 of file [Enum.cpp](#).



**8.8.3.5 set()** [1/2]

```
void SOIL::Enum::set (
    int value )
```

Set the currently selected item the enumeration holds based on its integer index.

**Exceptions**

<code>std::logic_error</code>	If the index exceeds the number of choices, an exception is thrown.
-------------------------------	---

**Parameters**

<code>in</code>	<code>value</code>	Index of the value to set
-----------------	--------------------	---------------------------

Definition at line 55 of file [Enum.cpp](#).

**8.8.3.6 set()** [2/2]

```
void SOIL::Enum::set (
    std::string value )
```

Set the currently selected item the enumeration holds based on a string value.

**Exceptions**

<code>std::logic_error</code>	If the value is not among the choices, an exception is thrown.
-------------------------------	--

**Parameters**

<code>in</code>	<code>value</code>	Value to set
-----------------	--------------------	--------------

Definition at line 67 of file [Enum.cpp](#).

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

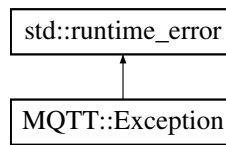
- [src/SOIL/Enum.h](#)
- [src/SOIL/Enum.cpp](#)

**8.9 MQTT::Exception Class Reference**

[MQTT Publisher Exception](#).

```
#include <LocalException.h>
```

Inheritance diagram for MQTT::Exception:



## Public Member Functions

- [Exception](#) (const char \*message="", int [code](#)=0)  
*Constructor.*
- [Exception](#) (const std::exception &exc)  
*Constructor from exception.*
- [~Exception](#) (void)  
*Destructor.*
- const int [code](#) (void) const  
*Result Code.*

### 8.9.1 Detailed Description

Specific subclass of `std::runtime_error` to distinguish exceptions coming from the [MQTT Publisher](#).

Definition at line 13 of file [LocalException.h](#).

### 8.9.2 Constructor & Destructor Documentation

#### 8.9.2.1 Exception() [1/2]

```
MQTT::Exception::Exception (
    const char * message = "",
    int code = 0 )
```

Constructor building an exception from a message and result code.

##### Parameters

in	<i>message</i>	Message to use for exception description
in	<i>code</i>	result code to store.

Definition at line 5 of file [LocalException.cpp](#).

### 8.9.2.2 Exception() [2/2]

```
MQTT::Exception::Exception (
    const std::exception & exc )
```

Constructor to rethrow an exception that has occurred from some other error within the context of the [MQTT Publisher](#).

#### Parameters

in	exc	Exception to replicate
----	-----	------------------------

Definition at line 9 of file [LocalException.cpp](#).

### 8.9.2.3 ~Exception()

```
MQTT::Exception::~~Exception (
    void )
```

Standard destructor for the class

Definition at line 14 of file [LocalException.cpp](#).

## 8.9.3 Member Function Documentation

### 8.9.3.1 code()

```
const int MQTT::Exception::code (
    void ) const [inline]
```

Get the result code that is stored internally in the exception.

#### Returns

result code.

Definition at line 55 of file [LocalException.h](#).

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

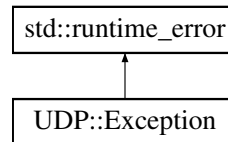
- [src/MQTT/LocalException.h](#)
- [src/MQTT/LocalException.cpp](#)

## 8.10 UDP::Exception Class Reference

[UDP Broadcast Exception](#).

```
#include <Exception.h>
```

Inheritance diagram for UDP::Exception:



### Public Member Functions

- [Exception](#) (const char \*message="")  
*Constructor.*
- [~Exception](#) (void)  
*Destructor.*

#### 8.10.1 Detailed Description

Specific subclass of `std::runtime_error` to distinguish exceptions coming from the UDO [Broadcast](#).

Definition at line 11 of file [Exception.h](#).

#### 8.10.2 Constructor & Destructor Documentation

##### 8.10.2.1 Exception()

```
UDP::Exception::Exception (
    const char * message = "" )
```

Constructor taking a message that is passed to `std::runtime_error`.

Parameters

in	<i>message</i>	Message to add to the runtime error
----	----------------	-------------------------------------

Definition at line 3 of file [Exception.cpp](#).

## 8.10.2.2 ~Exception()

```
UDP::Exception::~~Exception (
    void )
```

Default destructor, no special effort here.

Definition at line 7 of file [Exception.cpp](#).

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

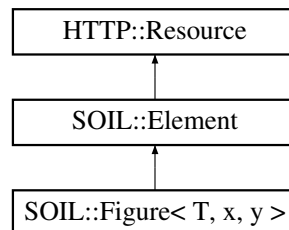
- [src/UDP/Exception.h](#)
- [src/UDP/Exception.cpp](#)

## 8.11 SOIL::Figure&lt; T, x, y &gt; Class Template Reference

Intermediate class for [Variable](#) and [Parameter](#) that derives from [Element](#).

```
#include <Figure.h>
```

Inheritance diagram for SOIL::Figure< T, x, y >:



## Public Member Functions

- [Figure](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [unit](#), std::string [ontology](#)="", [Range](#)< T > [range](#)=[Range](#)< T >(), [TIME](#) [time](#)=[TIME](#)())  
*Constructor.*
- [~Figure](#) ()  
*Destructor.*
- [Figure](#)< T, x, y > & [operator=](#) (const [Container](#)< T, x, y > &[value](#))  
*Assignment operator.*
- [Container](#)< T, x, y > & [operator\\*](#) (void)  
*Access Operator.*
- bool [check\\_range](#) (const [Container](#)< T, x, y > &[value](#)) const  
*Check range.*
- void [set\\_range](#) ([Range](#)< T > [range](#))  
*Set Range.*
- void [set\\_time](#) ([TIME](#) [time](#))  
*Set Time.*
- void [set\\_value](#) (const [Container](#)< T, x, y > &[value](#))  
*Set Value.*
- [HTTP::Json](#) [wjson](#) (void) override  
*HTTP JSON.*
- [Container](#)< T, x, y > [cast](#) (T [value](#))  
*Cast to container.*
- virtual void [update](#) (const [Container](#)< T, x, y > &[value](#), [TIME](#) [time](#))  
*Update.*

## Protected Member Functions

- virtual void [read](#) (void)=0  
*Read callback.*
- virtual void [write](#) (void)=0  
*Write callback.*

## Protected Attributes

- [TIME](#) [time](#)  
*Data Timestamp.*
- std::string [unit](#)  
*Unit.*
- [Container](#)< T, x, y > [value](#)  
*Value.*
- [Range](#)< T > [range](#)  
*Range.*

## Additional Inherited Members

### 8.11.1 Detailed Description

```
template<typename T, int x = -1, int y = -1>
class SOIL::Figure< T, x, y >
```

Intermediate class for [Variable](#) and [Parameter](#) that derives from [Element](#) as both share many properties. This class should not be instantiated directly and is abstract. The underlying data management completely relies on the templated [Container](#) class, hence many templates are passed on.

#### Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

Definition at line 48 of file [Figure.h](#).

### 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 Figure()

```
template<typename T , int x, int y>
SOIL::Figure< T, x, y >::Figure (
    std::shared_ptr< Element > parent,
```

```

std::string uuid,
std::string name,
std::string description,
std::string unit,
std::string ontology = "",
Range< T > range = Range<T>(),
SOIL::TIME time = TIME() )

```

Standard constructor initializing the values which should be called from the constructor of deriving classes.

#### Parameters

in	<i>parent</i>	Shared pointer to parent object.
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed
	<i>range</i>	[in] Allowed range for the variable values, defaults to an empty object, i.e. allowing all values
	<i>time</i>	[in] Timestamp for the initial value, defaults to unset

Definition at line 218 of file [Figure.h](#).

#### 8.11.2.2 ~Figure()

```

template<typename T , int x, int y>
SOIL::Figure< T, x, y >::~~Figure

```

Default destructor, without custom effort.

Definition at line 223 of file [Figure.h](#).

### 8.11.3 Member Function Documentation

#### 8.11.3.1 cast()

```

template<typename T , int x, int y>
SOIL::Container< T, x, y > SOIL::Figure< T, x, y >::cast (
    T value )

```

Takes a value and returns the corresponding container

#### Parameters

in	<i>value</i>	Value to containerize
----	--------------	-----------------------

**Returns**

[Container](#) version

**Todo** This function seems error prone in the multidimensional case.

Definition at line 248 of file [Figure.h](#).

**8.11.3.2 check\_range()**

```
template<typename T , int x, int y>
bool SOIL::Figure< T, x, y >::check_range (
    const Container< T, x, y > & value ) const
```

Check if value expressed as container object matches the range specified for this [Figure](#). This may be useful prior to assignment to avoid exceptions.

**Parameters**

<i>in</i>	<i>value</i>	Value to check
-----------	--------------	----------------

**Returns**

True if the value(s) is (are) in range, false else.

Definition at line 278 of file [Figure.h](#).

**8.11.3.3 operator\*()**

```
template<typename T , int x, int y>
SOIL::Container< T, x, y > & SOIL::Figure< T, x, y >::operator* (
    void )
```

Access Operator returning the container of the value.

**Returns**

Value container.

Definition at line 271 of file [Figure.h](#).

**8.11.3.4 operator=()**

```
template<typename T , int x, int y>
SOIL::Figure< T, x, y > & SOIL::Figure< T, x, y >::operator= (
    const Container< T, x, y > & value )
```

Assigns the value provided as container on the right hand side to the figure.



## Exceptions

<code>std::range_error</code>	Throws an exception if the value to assign is outside the allowed range.
-------------------------------	--

## Parameters

<code>in</code>	<code>value</code>	Value to assign
-----------------	--------------------	-----------------

## Returns

Reference to the current [Figure](#)

Definition at line 259 of file [Figure.h](#).

## 8.11.3.5 read()

```
template<typename T , int x, int y>
void SOIL::Figure< T, x, y >::read (
    void ) [protected], [pure virtual]
```

Read callback that can be implemented by deriving classes to perform custom build logic on read actions, e.g. update the value from an external storage. Declared virtual to make sure the derived method is called first.

Implemented in [SOIL::Parameter< T, x, y >](#), and [SOIL::Variable< T, x, y >](#).

Definition at line 254 of file [Figure.h](#).

## 8.11.3.6 set\_range()

```
template<typename T , int x, int y>
void SOIL::Figure< T, x, y >::set_range (
    Range< T > range )
```

Set the range property of this figure.

## Parameters

<code>in</code>	<code>range</code>	<a href="#">Range</a> to set
-----------------	--------------------	------------------------------

Definition at line 284 of file [Figure.h](#).

## 8.11.3.7 set\_time()

```
template<typename T , int x, int y>
```

```
void SOIL::Figure< T, x, y >::set_time (
    TIME time )
```

Set the time property of this figure.

#### Parameters

in	<i>time</i>	Time to set
----	-------------	-------------

Definition at line 291 of file [Figure.h](#).

### 8.11.3.8 set\_value()

```
template<typename T , int x, int y>
void SOIL::Figure< T, x, y >::set_value (
    const Container< T, x, y > & value )
```

Assigns the value provided as container. This is currently equivalent to the assignment operator.

#### Exceptions

<code>std::range_error</code>	Throws an exception if the value to assign is outside the allowed range.
-------------------------------	--

#### Parameters

in	<i>value</i>	Value to assign
----	--------------	-----------------

Definition at line 298 of file [Figure.h](#).

### 8.11.3.9 update()

```
template<typename T , int x, int y>
void SOIL::Figure< T, x, y >::update (
    const Container< T, x, y > & value,
    TIME time ) [virtual]
```

Update the figure setting a new value and timestamp. This is common scenario when dealing with measurement data.

#### Parameters

in	<i>value</i>	Value to set
in	<i>time</i>	Timestamp to assign

Definition at line 308 of file [Figure.h](#).

### 8.11.3.10 wjson()

```
template<typename T , int x, int y>
HTTP::Json SOIL::Figure< T, x, y >::wjson (
    void ) [override], [virtual]
```

Get a [HTTP](#) JSON object corresponding to the current state of the [Figure](#). This function provides a partial representation of the SOIL-conformant JSON representation of [Variable](#) and [Parameter](#) and may be called from their [wjson\(\)](#) methods.

#### Returns

JSON object

Reimplemented from [SOIL::Element](#).

Definition at line 229 of file [Figure.h](#).

### 8.11.3.11 write()

```
template<typename T , int x = -1, int y = -1>
virtual void SOIL::Figure< T, x, y >::write (
    void ) [protected], [pure virtual]
```

Write callback that can be implemented by deriving classes to perform custom build logic on write actions, e.g.set a value to an external system. Declared virtual to make sure the derived method is called first.

Implemented in [SOIL::Parameter< T, x, y >](#), and [SOIL::Variable< T, x, y >](#).

## 8.11.4 Member Data Documentation

### 8.11.4.1 range

```
template<typename T , int x = -1, int y = -1>
Range<T> SOIL::Figure< T, x, y >::range [protected]
```

Allowed range for the figure, expressed using the therefore designed [Range](#) class.

Definition at line 77 of file [Figure.h](#).

#### 8.11.4.2 time

```
template<typename T , int x = -1, int y = -1>
TIME SOIL::Figure< T, x, y >::time [protected]
```

Timestamp of the data, i.e. the time which should be considered as physically related to the value.

Definition at line 56 of file [Figure.h](#).

#### 8.11.4.3 unit

```
template<typename T , int x = -1, int y = -1>
std::string SOIL::Figure< T, x, y >::unit [protected]
```

Unit of the stored value, expressed as UNECE code (e.g. MTR).

Definition at line 63 of file [Figure.h](#).

#### 8.11.4.4 value

```
template<typename T , int x = -1, int y = -1>
Container<T, x, y> SOIL::Figure< T, x, y >::value [protected]
```

Actual value that is currently held by the figure, which is represented as container.

Definition at line 70 of file [Figure.h](#).

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

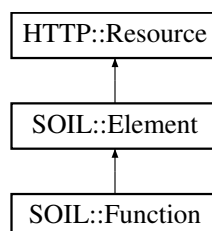
- [src/SOIL/Figure.h](#)

## 8.12 SOIL::Function Class Reference

[Function](#) Class.

```
#include <Function.h>
```

Inheritance diagram for SOIL::Function:



## Public Member Functions

- DLL [Function](#) (std::shared\_ptr< [Element](#) > parent, std::string uuid, std::string name, std::string description, std::string ontology="")  
*Constructor.*
- DLL [~Function](#) ()  
*Destructor.*
- template<typename T , int x = -1, int y = -1>  
void [add\\_argument](#) (std::string uuid, std::string name, std::string description, std::string unit, [SOIL::Range](#)< T > range=[SOIL::Range](#)< T >(), std::string ontology="")  
*Add argument.*
- template<typename T , int x = -1, int y = -1>  
void [add\\_argument](#) (std::string uuid, std::string name, std::string description, std::string unit, [SOIL::Range](#)< T > range, const [Container](#)< T, x, y > &default\_value, std::string ontology="")  
*Add argument with default value.*
- template<typename T , int x = -1, int y = -1>  
void [add\\_return](#) (std::string uuid, std::string name, std::string description, std::string unit, [SOIL::Range](#)< T > range=[SOIL::Range](#)< T >(), std::string ontology="")  
*Add return value.*
- template<typename T , int x = -1, int y = -1>  
[HTTP::Json](#) [make\\_return](#) (std::string uuid, [Container](#)< T, x, y > value)  
*Make JSON Return.*
- template<typename T , int x = -1, int y = -1>  
[Container](#)< T, x, y > [make\\_argument](#) (std::string uuid, [HTTP::Json](#) external\_json)  
*Make Argument from JSON.*
- DLL [HTTP::Json](#) [wjson](#) (void)  
*HTTP JSON.*
- virtual DLL [HTTP::Response](#) [handle\\_get](#) ([HTTP::Request](#) request, std::smatch match=std::smatch())  
*Handle HTTP GET request.*
- virtual DLL [HTTP::Response](#) [handle\\_post](#) ([HTTP::Request](#) request, std::smatch match=std::smatch())  
*Handle HTTP POST request.*
- virtual DLL std::map< std::string, [HTTP::Json](#) > [invoke](#) (std::map< std::string, [HTTP::Json](#) > arguments)  
*Core Invocation.*
- DLL std::shared\_ptr< [Function](#) > [ptr](#) (void)  
*Get Pointer.*

## Static Public Member Functions

- static std::shared\_ptr< [Function](#) > [create](#) (std::shared\_ptr< [Element](#) > parent, std::string uuid, std::string name, std::string description, std::string ontology="")  
*Create new Function.*

## Additional Inherited Members

### 8.12.1 Detailed Description

Class encapsulating a function represented in [SOIL](#). The class directly inherits from [Element](#). The recommended development pattern is to subclass this class and instantiate a use-case specific version. In this case, the [invoke\(\)](#) method should be overridden to implement the function's business logic.

Arguments are represented using the [Parameter](#) class. A [Function](#) supports the [HTTP](#) GET (to introspect the signature) and [HTTP](#) POST (to execute the function) verbs.

Definition at line 21 of file [Function.h](#).

## 8.12.2 Constructor & Destructor Documentation

### 8.12.2.1 Function()

```
SOIL::Function::Function (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string ontology = "" )
```

Constructor that shall be used to instantiate a [Function](#). If the class is subclassed, it should be called from the respective constructor. Most parameters are passed on to the constructor of [Element](#).

#### Parameters

in	<i>parent</i>	Shared pointer to parent object. Can be set to NULL for the creation of a root object
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed

Definition at line 4 of file [Function.cpp](#).

### 8.12.2.2 ~Function()

```
SOIL::Function::~~Function ( )
```

Standard destructor

Definition at line 14 of file [Function.cpp](#).

## 8.12.3 Member Function Documentation

### 8.12.3.1 add\_argument() [1/2]

```
template<typename T , int x, int y>
void SOIL::Function::add_argument (
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    SOIL::Range< T > range,
    const Container< T, x, y > & default_value,
    std::string ontology = "" )
```

Add an argument when building the function with a default value. This method should typically be called from the subclass constructor. It uses the same templating system than [Container](#) does to handle multidimensional data.

## Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

## Parameters

in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>range</i>	Allowed range for this argument, which is checked prior to execution. Defaults to no range restriction.
in	<i>default</i>	Default value provided as <a href="#">Container</a>
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed (default)

Definition at line 234 of file [Function.h](#).

8.12.3.2 `add_argument()` [2/2]

```
template<typename T , int x, int y>
void SOIL::Function::add_argument (
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    SOIL::Range< T > range = SOIL::Range<T> (),
    std::string ontology = "" )
```

Add an argument when building the function. This method should typically be called from the subclass constructor. It uses the same templating system than [Container](#) does to handle multidimensional data.

## Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

## Parameters

in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>range</i>	Allowed range for this argument, which is checked prior to execution. Defaults to no range restriction.
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed (default)

Definition at line 229 of file [Function.h](#).

### 8.12.3.3 add\_return()

```
template<typename T , int x, int y>
void SOIL::Function::add_return (
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    SOIL::Range< T > range = SOIL::Range<T>(),
    std::string ontology = "" )
```

Add areturn value when building the function. This method should typically be called from the subclass constructor. It uses the same templating system than [Container](#) does to handle multidimensional data.

#### Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitrary size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitrary size. Must be -1 for <i>x</i> to be -1.

#### Parameters

in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>range</i>	Allowed range for this argument. Defaults to no range restriction.
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed (default)

Definition at line 243 of file [Function.h](#).

### 8.12.3.4 create()

```
std::shared_ptr< SOIL::Function > SOIL::Function::create (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string ontology = "" ) [static]
```

Create a new [Function](#) using the default constructor and return a shared pointer reference. This is the preferred method for manual creation with consistent lifecycle handling.



## Parameters

in	<i>parent</i>	Shared pointer to parent object. Can be set to NULL for the creation of a root object
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed

Definition at line 18 of file [Function.cpp](#).

8.12.3.5 `handle_get()`

```
HTTP::Response SOIL::Function::handle_get (
    HTTP::Request request,
    std::smatch match = std::smatch() ) [virtual]
```

On [HTTP](#) GET, the function returns information about itself and its signature. It is not invoked and this call should not lead to side effects.

## Parameters

in	<i>request</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 46 of file [Function.cpp](#).

8.12.3.6 `handle_post()`

```
HTTP::Response SOIL::Function::handle_post (
    HTTP::Request request,
    std::smatch match = std::smatch() ) [virtual]
```

On [HTTP](#) POST, the function is invoked. With the POST request, a list of arguments must be passed in the request's body as list under the keyword "arguments". This method calls [invoke\(\)](#). If you do not implement this function, nothing will happen.

## Parameters

in	<i>request</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestdsk

Reimplemented from [HTTP::Resource](#).

Definition at line 54 of file [Function.cpp](#).

**8.12.3.7 invoke()**

```
std::map< std::string, HTTP::Json > SOIL::Function::invoke (
    std::map< std::string, HTTP::Json > arguments ) [virtual]
```

This is the main function that **MUST** be implemented in a derived class to meaningful act as a function. The arguments are passed as raw [HTTP::Json](#) objects mapped to their UUIDs. A map of [HTTP::Json](#) objects is expected as returns. It is recommend to use `make_arguments` and `make_returns` in this function. Arguments and returns are not further decomposed to Containers to support multi-typed values.

**Postcondition**

This function will be called from the [HTTP](#) POST handler. Any exception thrown will be caught there again and lead to an [HTTP](#) Internal Error (500) return code.

**Exceptions**

<code>std::logic_error</code>	If not implemented in the derived class, the implementation here throws an exception.
-------------------------------	---

**Parameters**

in	<i>arguments</i>	Map of arguments as <a href="#">HTTP</a> Json objects indexrd by their UUID
----	------------------	---

**Returns**

Map of return values as [HTTP](#) Json objects indexrd by their UUID

Definition at line 95 of file [Function.cpp](#).

**8.12.3.8 make\_argument()**

```
template<typename T , int x, int y>
Container< T, x, y > SOIL::Function::make_argument (
    std::string uuid,
    HTTP::Json external_json )
```

Helper function that converts a [HTTP](#) Json object to a container of the passed value This function is usually useful in the custom implementation of [invoke\(\)](#).

## Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

## Parameters

in	<i>uuid</i>	Locally Unique identifier of return value
in	<i>external_json</i>	JSON object from which to convert

## Returns

Value as [Container](#)

Definition at line 256 of file [Function.h](#).

## 8.12.3.9 make\_return()

```
template<typename T , int x, int y>
HTTP::Json SOIL::Function::make_return (
    std::string uuid,
    Container< T, x, y > value )
```

Helper function that converts a return value provided as [Container](#) to a correpsonding JSON object. This function is usually useful in the custom implementation of [invoke\(\)](#).

## Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitray size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitray size. Must be -1 for <i>x</i> to be -1.

## Parameters

in	<i>uuid</i>	Locally Unique identifier of return value
in	<i>value</i>	Value of the return value

## Returns

JSON object corresponding to [SOIL](#) representation

Definition at line 248 of file [Function.h](#).

### 8.12.3.10 ptr()

```
DLL std::shared_ptr< Function > SOIL::Function::ptr (
    void ) [inline]
```

Return a shared pointer casted to the [Function](#) type to element itself.

#### Returns

Casted pointer

Definition at line 223 of file [Function.h](#).

### 8.12.3.11 wjson()

```
HTTP::Json SOIL::Function::wjson (
    void ) [virtual]
```

Get a [HTTP](#) JSON object corresponding to the function including arguments and returns

#### Returns

JSON object

Reimplemented from [SOIL::Element](#).

Definition at line 24 of file [Function.cpp](#).

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

- [src/SOIL/Function.h](#)
- [src/SOIL/Function.cpp](#)

## 8.13 SIGN::Hasher Class Reference

SHA256 [Hasher](#).

```
#include <Hasher.h>
```

### Public Member Functions

- [Hasher](#) ()  
*Constructor.*
- [~Hasher](#) ()  
*Destructor.*
- [template<typename T > void push\\_back \(T x\)](#)  
*Add data.*
- [std::vector< unsigned char > hash \(\)](#)  
*Hash the data buffer.*
- [void reset \(\)](#)  
*Reset data buffer.*
- [size\\_t size](#) (void)  
*Digest size.*

## Static Public Member Functions

- static std::string [print](#) (std::vector< unsigned char > bytes, bool uppercase=true)  
*Print Bytestring.*
- static std::vector< unsigned char > [sha256](#) (const unsigned char \*data, size\_t length)  
*SHA256 hash.*

### 8.13.1 Detailed Description

Class which provides an convenient interface for calculating SHA256 hashes. The underlying methods are taken from OpenSSL.

Definition at line 14 of file [Hasher.h](#).

### 8.13.2 Constructor & Destructor Documentation

#### 8.13.2.1 Hasher()

```
SIGN::Hasher::Hasher ( )
```

Default constructor, no special effort here.

Definition at line 19 of file [Hasher.cpp](#).

#### 8.13.2.2 ~Hasher()

```
SIGN::Hasher::~Hasher ( )
```

Default destructor, no special effort here.

Definition at line 24 of file [Hasher.cpp](#).

### 8.13.3 Member Function Documentation

### 8.13.3.1 hash()

```
std::vector< unsigned char > SIGN::Hasher::hash ( )
```

Calculate the SHA256 hash of the current data buffer and return it as standard vector of bytes. Internally calls the static SHA256 function

#### Postcondition

The output of this function can be used in the signer.

#### Returns

Hash result as sequenxe of bytes.

Definition at line 28 of file [Hasher.cpp](#).

### 8.13.3.2 print()

```
std::string SIGN::Hasher::print (
    std::vector< unsigned char > bytes,
    bool uppercase = true ) [static]
```

Convenience function to pretty-print bytestrings in HEX format.

#### Parameters

in	<i>bytes</i>	Data to print
in	<i>uppercase</i>	Boolean flag whether to print uppercase

#### Returns

String with print result.

Definition at line 43 of file [Hasher.cpp](#).

### 8.13.3.3 push\_back()

```
template<typename T >
void SIGN::Hasher::push_back (
    T x ) [inline]
```

Push back data to the internal buffer which eventually gets hashed. This function is implemented using templates, underneath it copies the raw bytes after casting.

## Template Parameters

<i>T</i>	Original type of the item to add to the buffer
----------	--

## Parameters

<i>in</i>	<i>x</i>	Item to add to the buffer
-----------	----------	---------------------------

Definition at line 99 of file [Hasher.h](#).

### 8.13.3.4 reset()

```
void SIGN::Hasher::reset ( )
```

Clear the internal data buffer (i.e. to start with new data)

Definition at line 33 of file [Hasher.cpp](#).

### 8.13.3.5 sha256()

```
std::vector< unsigned char > SIGN::Hasher::sha256 (
    const unsigned char * data,
    size_t length ) [static]
```

Calculate the SHA256 hash of the given input data. Be careful to no pass invalid pointers here, there is a risk of memory leaks.

## Parameters

<i>data</i>	[in] Pointer to the memory block to read from
<i>length</i>	[in] Length of the data to consume

## Returns

SHA256 hash as vector of bytes

Definition at line 8 of file [Hasher.cpp](#).

### 8.13.3.6 size()

```
size_t SIGN::Hasher::size (
    void )
```

Convenience function to get the digest size, which is often needed when handling raw bytes.

#### Returns

SHA256 digest length

Definition at line 38 of file [Hasher.cpp](#).

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

- src/SIGN/[Hasher.h](#)
- src/SIGN/[Hasher.cpp](#)

## 8.14 MQTT::MessageContainer Struct Reference

Internal [MQTT](#) message struct.

```
#include <MessageContainer.h>
```

### Public Attributes

- `std::string` [topic](#)  
*Message Topic.*
- `std::string` [message](#)  
*Message.*
- `int` [qos](#)  
*Quality of Service.*
- `bool` [retain](#)  
*Retain Flag.*

### 8.14.1 Detailed Description

Struct to manage [MQTT](#) message data internally using standard strings.

Definition at line 10 of file [MessageContainer.h](#).

### 8.14.2 Member Data Documentation

#### 8.14.2.1 message

```
std::string MQTT::MessageContainer::message
```

Primary content of the message to send.

Definition at line 24 of file [MessageContainer.h](#).



### 8.14.2.2 qos

```
int MQTT::MessageContainer::qos
```

[MQTT](#) Quality of service level to choose for the message. Check the [MQTT](#) specifications for the exact behaviours of 0, 1, and 2 in conjunction with message retention.

Definition at line 32 of file [MessageContainer.h](#).

### 8.14.2.3 retain

```
bool MQTT::MessageContainer::retain
```

Flag whether to retain the message on the broker after disconnection.

Definition at line 39 of file [MessageContainer.h](#).

### 8.14.2.4 topic

```
std::string MQTT::MessageContainer::topic
```

Topic to which the message shall be sent.

Definition at line 17 of file [MessageContainer.h](#).

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

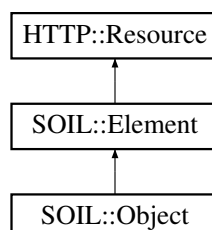
- [src/MQTT/MessageContainer.h](#)

## 8.15 SOIL::Object Class Reference

[Object](#) Class.

```
#include <Object.h>
```

Inheritance diagram for SOIL::Object:



## Public Member Functions

- [Object](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [ontology](#)="")  
*Constructor.*
- [~Object](#) ()  
*Destructor.*
- web::json::value [wjson](#) (void)  
*HTTP JSON.*
- virtual void [read](#) (void)  
*Read callback.*
- virtual [HTTP::Json](#) [insert](#) ([HTTP::Json](#) body)  
*Insert callback.*
- virtual void [remove](#) (void)  
*Remove callback.*
- [HTTP::Response](#) [handle\\_get](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP GET Handler.*
- [HTTP::Response](#) [handle\\_put](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP PUT Handler.*
- [HTTP::Response](#) [handle\\_delete](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP DELETE Handler.*
- std::shared\_ptr< [Object](#) > [ptr](#) (void)  
*Get Pointer.*

## Static Public Member Functions

- static std::shared\_ptr< [Object](#) > [create](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [ontology](#)="")  
*Create new [Object](#).*

## Additional Inherited Members

### 8.15.1 Detailed Description

Class implementing the [SOIL Object](#) type. The purpose of Objects is mainly to organize the overall system. Only Objects can have child items, which can be added or removed during runtime. Objects support [HTTP](#) GET (read), PUT (add child) and DELETE (remove child) verbs. This class directly inherits from [Element](#).

**Todo** The [HTTP](#) handlers may be moved to protected if [HTTP::Server](#) is declared as friend class. Currently this is not done to allow for greater flexibility.

Definition at line 18 of file [Object.h](#).

### 8.15.2 Constructor & Destructor Documentation

### 8.15.2.1 Object()

```
SOIL::Object::Object (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string ontology = "" )
```

Default constructor for [SOIL](#) elements, to be called from derived classes.

#### Parameters

in	<i>parent</i>	Shared pointer to parent object. Can be set to NULL for the creation of a root object
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed

#### Precondition

The ontology referred to must be defined elsewhere and is decoupled from the code.

Definition at line 4 of file [Object.cpp](#).

### 8.15.2.2 ~Object()

```
SOIL::Object::~~Object ( )
```

Standard destructor. The main destruction occurs in the destructor of [Element](#), i.e. the references to the child elements are deleted, e.g. they go out of scope if this was the last shared pointer.

Definition at line 14 of file [Object.cpp](#).

## 8.15.3 Member Function Documentation

### 8.15.3.1 create()

```
std::shared_ptr< SOIL::Object > SOIL::Object::create (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string ontology = "" ) [static]
```

Create a new [Object](#) using the default constructor and return a shared pointer reference. This is the preferred method for manual creation with consistent lifecycle handling.

**Parameters**

in	<i>parent</i>	Shared pointer to parent object. Can be set to NULL for the creation of a root object
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed

**Precondition**

The ontology referred to must be defined elsewhere and is decoupled from the code.

Definition at line 18 of file [Object.cpp](#).

**8.15.3.2 handle\_delete()**

```
HTTP::Response SOIL::Object::handle_delete (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a DELETE method. This function deletes an element. It should not be overridden directly in subclasses, instead the [remove\(\)](#) function should be overridden. It is mandatory to implement a custom logic in the aforementioned function to allow PUT to properly work. If manual deletion is not foreseen, you may consider overriding `allowed_methods` without DELETE.

If no exception is thrown, an empty response with result code OK is returned, otherwise an Internal Error will be provided.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by <code>cpprestsdk</code>
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by `cpprestsdk`

Reimplemented from [HTTP::Resource](#).

Definition at line 75 of file [Object.cpp](#).

**8.15.3.3 handle\_get()**

```
HTTP::Response SOIL::Object::handle_get (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a GET method. This function returns a representation of the [Object](#) to the requesting party. It should not be overridden directly in subclasses, instead the [read\(\)](#) function should be overridden.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 54 of file [Object.cpp](#).

**8.15.3.4 handle\_put()**

```
HTTP::Response SOIL::Object::handle_put (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a PUT method. This function inserts an element to the objects children. It should not be overridden directly in subclasses, instead the [HTTP::Json insert\(HTTP::Json body\)](#) function should be overridden. It is mandatory to implement a custom logic in the aforementioned function to allow PUT to properly work. If manual insertion is not foreseen, you may consider overriding `allowed_methods` without PUT. The minimum requirement to the body is that it contains an valid UUID key with value, the templates and further logic may be implemented by server.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 63 of file [Object.cpp](#).

**8.15.3.5 insert()**

```
HTTP::Json SOIL::Object::insert (
    HTTP::Json body ) [virtual]
```

**Function** that is called when a new child object is insterted via [HTTP](#) PUT. This function muss be implemented by the deriving class.

**Exceptions**

<code>std::logic_error</code>	If the derived class does not contain an implementation, an exception is thrown here. The exception will be caught in the <a href="#">HTTP</a> handler of the server and returns with an <a href="#">HTTP</a> Internal Error (500) code.
-------------------------------	--

**Parameters**

<code>in</code>	<code>body</code>	<a href="#">HTTP</a> JSON body passed to PUT
-----------------	-------------------	--

**Returns**

[HTTP](#) JSON object corresponding to the created [Object](#), as required per good [HTTP](#) practice.

Definition at line [44](#) of file [Object.cpp](#).

**8.15.3.6 ptr()**

```
std::shared_ptr< Object > SOIL::Object::ptr (
    void ) [inline]
```

Return a shared pointer casted to the [Object](#) type to element itself.

**Returns**

Casted pointer

Definition at line [149](#) of file [Object.h](#).

**8.15.3.7 read()**

```
void SOIL::Object::read (
    void ) [virtual]
```

Read callback that can be implemented by deriving classes to perform custom build logic on read actions, e.g. update the value from an external storage. Declared virtual to make sure the derived method is called first. Any exception that may be occurring will be caught in the [HTTP](#) handler.

Definition at line [40](#) of file [Object.cpp](#).

### 8.15.3.8 remove()

```
void SOIL::Object::remove (
    void ) [virtual]
```

Callback hook for subclasses to add custom business logic to the [HTTP](#) DELETE operation, e.g. to act on external resources. Otherwise the object will just be deleted.

Definition at line 49 of file [Object.cpp](#).

### 8.15.3.9 wjson()

```
HTTP::Json SOIL::Object::wjson (
    void ) [virtual]
```

Get a [HTTP](#) JSON object corresponding to the function including arguments and returns

#### Returns

JSON object

Reimplemented from [SOIL::Element](#).

Definition at line 24 of file [Object.cpp](#).

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

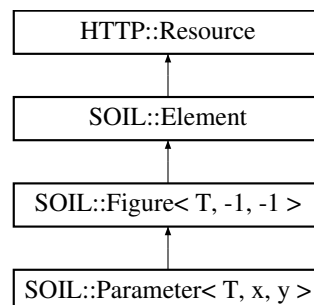
- [src/SOIL/Object.h](#)
- [src/SOIL/Object.cpp](#)

## 8.16 SOIL::Parameter< T, x, y > Class Template Reference

[Parameter](#) Class.

```
#include <Parameter.h>
```

Inheritance diagram for SOIL::Parameter< T, x, y >:



## Public Member Functions

- [Parameter](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [unit](#), bool [constant](#)=false, std::string [ontology](#)="", [Range](#)< T > [range](#)=[Range](#)< T >(), [TIME](#) [time](#)=[TIME](#)())  
*Constructor.*
- [~Parameter](#) ()  
*Destructor.*
- [Parameter](#)< T, x, y > & [operator=](#) (const [Container](#)< T, x, y > &[value](#))  
*Assignment operator.*
- [HTTP::Json wjson](#) (void) override  
*HTTP JSON.*
- [HTTP::Response handle\\_get](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP GET Handler.*
- [HTTP::Response handle\\_patch](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP PATCH Handler.*
- std::shared\_ptr< [Parameter](#) > [ptr](#) (void)  
*Get Pointer.*
- bool [mqtt](#) (std::shared\_ptr< [MQTT::Publisher](#) > publisher, int qos=0, bool retain=false)  
*Publish to MQTT.*

## Static Public Member Functions

- static std::shared\_ptr< [Parameter](#) > [create](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [unit](#), bool [constant](#)=false, std::string [ontology](#)="", [Range](#)< T > [range](#)=[Range](#)< T >(), [TIME](#) [time](#)=[TIME](#)())  
*Create new Parameter.*

## Protected Member Functions

- virtual void [read](#) (void)  
*Read callback.*
- virtual void [write](#) (void)  
*Write callback.*

## Protected Attributes

- bool [constant](#)  
*Constant flag.*



## Additional Inherited Members

### 8.16.1 Detailed Description

```
template<typename T, int x = -1, int y = -1>
class SOIL::Parameter< T, x, y >
```

This class represents a [SOIL Parameter](#). [Parameter](#) and [Variable](#) share many common properties and therefore both inherit from [Figure](#), such that methods there should be considered in any case. Parameters are primarily intended for values that do not represent any measurement or physical phenomenon outside the immediate control of device. In contrast to variables, they can be set externally to control the device and do not possess a measurement uncertainty. Parameter may be constant. In lightweight scenarios, this class may be instantiated directly, for more specific scenarios, it should be subclassed and override the implementations of [read\(\)](#) and [write\(\)](#).

A parameter supports [HTTP](#) GET (read) and [HTTP](#) PATCH (set) verbs.

The data management of [Figure](#) and hence [Parameter](#) relies on [Container](#), such that the same templating logic is used.

#### Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitrary size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitrary size. Must be -1 for <i>x</i> to be -1.

**Todo** The [HTTP](#) handlers may be moved to protected if [HTTP::Server](#) is declared as friend class. Currently this is not done to allow for greater flexibility.

Definition at line 34 of file [Parameter.h](#).

### 8.16.2 Constructor & Destructor Documentation

#### 8.16.2.1 Parameter()

```
template<typename T , int x = -1, int y = -1>
SOIL::Parameter< T, x, y >::Parameter (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    bool constant = false,
    std::string ontology = "",
    Range< T > range = Range<T>(),
    TIME time = TIME() )
```

Standard constructor initializing the values. If subclassed, it should be called from the subclass constructor.

## Parameters

in	<i>parent</i>	Shared pointer to parent object.
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>constant</i>	Boolean flag whether this value is constant
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed
in	<i>range</i>	Allowed range for the variable values, defaults to an empty object, i.e. allowing all values
in	<i>time</i>	Timestamp for the initial value, defaults to unset

Definition at line 202 of file [Parameter.h](#).

### 8.16.2.2 ~Parameter()

```
template<typename T , int x, int y>
SOIL::Parameter< T, x, y >::~~Parameter
```

Default destructor, without custom effort.

Definition at line 213 of file [Parameter.h](#).

## 8.16.3 Member Function Documentation

### 8.16.3.1 create()

```
template<typename T , int x, int y>
std::shared_ptr< SOIL::Parameter< T, x, y > > SOIL::Parameter< T, x, y >::create (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    bool constant = false,
    std::string ontology = "",
    Range< T > range = Range<T>(),
    TIME time = TIME() ) [inline], [static]
```

Create a new [Parameter](#) using the default constructor and return a shared pointer reference. This is the preferred method for manual creation with consistent lifecycle handling.

## Parameters

in	<i>parent</i>	Shared pointer to parent object.
in	<i>uuid</i>	Locally Unique identifier

## Parameters

in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>constant</i>	Boolean flag whether this value is constant
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed
in	<i>range</i>	Allowed range for the variable values, defaults to an empty object, i.e. allowing all values
in	<i>time</i>	Timestamp for the initial value, defaults to unset

Definition at line 218 of file [Parameter.h](#).

8.16.3.2 `handle_get()`

```
template<typename T , int x, int y>
HTTP::Response SOIL::Parameter< T, x, y >::handle_get (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [inline], [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a GET method. This function returns a representation of the [Parameter](#) and its current value to the requesting party. It should not be overridden directly in subclasses, instead the `read()` function should be overridden.

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cprestdsk
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cprestdsk

Reimplemented from [HTTP::Resource](#).

Definition at line 254 of file [Parameter.h](#).

8.16.3.3 `handle_patch()`

```
template<typename T , int x, int y>
HTTP::Response SOIL::Parameter< T, x, y >::handle_patch (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [inline], [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a PATCH method. This function updates the value of a parameter. The request must at least contain a It should not be overridden directly in subclasses, instead the `write()` function should be overridden.

If an update is not not foreseen, you may consider overriding `allowed_methods` without PATCH. This is not enforced by default through the `constant` flag. The minimum requirement to the body is that it contains a value to assign. If a timestamp is provided, it will be set, otherwise the current server time is taken.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 268 of file [Parameter.h](#).

**8.16.3.4 mqtt()**

```
template<typename T , int x, int y>
bool SOIL::Parameter< T, x, y >::mqtt (
    std::shared_ptr< MQTT::Publisher > publisher,
    int qos = 0,
    bool retain = false ) [inline]
```

Publish the current JSON representation under the FQID as topic using a given [MQTT](#) publisher. This function must be explicitly called from the user's code as otherwise the update cycle would depend on the publisher and the user will be left without control to call other methods before publishing.

A good pattern is to have a reference to an [MQTT](#) publisher in a subclassing implementation and implement a complete update cycle there.

**Precondition**

An [MQTT::Publisher](#) must be instantiated elsewhere and have a valid lifecycle.

**Parameters**

in	<i>publisher</i>	Reference to the publisher to use
in	<i>qos</i>	Quality of service to choose for <a href="#">MQTT</a> message
in	<i>retain</i>	Flag whether to retain the message on the server.

**Returns**

Boolean flag whether the message was accepted in the message queue.

**Todo** Currently the implementation of this function is redundant in [Variable](#) and [Parameter](#). It is deliberately not moved to [Figure](#) as different implementations may occur in the future, but would be a valid option.

Definition at line 300 of file [Parameter.h](#).

### 8.16.3.5 operator=()

```
template<typename T , int x, int y>
SOIL::Parameter< T, x, y > & SOIL::Parameter< T, x, y >::operator= (
    const Container< T, x, y > & value ) [inline]
```

Assigns the value provided as container on the right hand side to the parameter. Immediately resorts to the implementation in [Figure](#) internally.

#### Exceptions

<code>std::range_error</code>	Throws an exception if the value to assign is outside the allowed range.
-------------------------------	--

#### Parameters

<code>in</code>	<code>value</code>	Value to assign
-----------------	--------------------	-----------------

#### Returns

Reference to the current [Parameter](#)

Definition at line 225 of file [Parameter.h](#).

### 8.16.3.6 ptr()

```
template<typename T , int x = -1, int y = -1>
std::shared_ptr< Parameter > SOIL::Parameter< T, x, y >::ptr (
    void ) [inline]
```

Return a shared pointer casted to the [Parameter](#) type to element itself.

#### Returns

Casted pointer

Definition at line 166 of file [Parameter.h](#).

### 8.16.3.7 read()

```
template<typename T , int x, int y>
void SOIL::Parameter< T, x, y >::read (
    void ) [protected], [virtual]
```

Read callback that can be implemented by deriving classes to perform custom build logic on read actions, e.g. update the value from an external storage. Declared virtual to make sure the derived method is called first. Does nothing by default.

Implements [SOIL::Figure< T, -1, -1 >](#).

Definition at line 243 of file [Parameter.h](#).

### 8.16.3.8 wjson()

```
template<typename T , int x, int y>
HTTP::Json SOIL::Parameter< T, x, y >::wjson (
    void ) [override], [virtual]
```

Get a [HTTP](#) JSON object corresponding to the current state of the [Parameter](#). This function provides a SOIL-conformant JSON representation the parameter. It internally extends the method of [Figure](#).

#### Returns

JSON object

Reimplemented from [SOIL::Element](#).

Definition at line [233](#) of file [Parameter.h](#).

### 8.16.3.9 write()

```
template<typename T , int x, int y>
void SOIL::Parameter< T, x, y >::write (
    void ) [inline], [protected], [virtual]
```

Write callback that can be implemented by deriving classes to perform custom build logic on write actions, e.g.set a value to an external system. Declared virtual to make sure the derived method is called first. Does nothing by default.

Implements [SOIL::Figure< T, -1, -1 >](#).

Definition at line [248](#) of file [Parameter.h](#).

## 8.16.4 Member Data Documentation

### 8.16.4.1 constant

```
template<typename T , int x = -1, int y = -1>
bool SOIL::Parameter< T, x, y >::constant [protected]
```

Boolean flag whether this parameter should be considered as constant.

Definition at line [46](#) of file [Parameter.h](#).

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

- [src/SOIL/Parameter.h](#)

## 8.17 MQTT::Publisher Class Reference

[MQTT Publisher](#).

```
#include <Publisher.h>
```

### Public Member Functions

- [Publisher](#) (std::string id, unsigned int buffer=1024)  
*Constructor.*
- [~Publisher](#) ()  
*Destructor.*
- bool [publish](#) (std::string topic, std::string message, int qos, bool retain)  
*Publish an [MQTT](#) message.*
- bool [publish](#) (std::vector< std::string > topics, std::vector< std::string > messages, int qos, bool retain)  
*Publish multiple [MQTT](#) messages.*
- void [connect](#) ([MQTT::Configuration](#) configuration)  
*Connect to broker.*
- void [connect](#) ()  
*Connect to broker.*
- void [reconnect](#) (void)  
*Direct Reconnect.*
- void [disconnect](#) (unsigned int timeout=10000)  
*Disconnect from the broker.*
- void [set\\_root\\_topic](#) (std::string root\_topic)  
*Set new root topic.*
- void [set\\_buffer](#) (unsigned int buffer)  
*Set message queue size.*
- bool [is\\_connected](#) (void)  
*Is connected?*
- int [min\\_delay\\_ms](#) (void)  
*Minimum delay between to messages in milliseconds.*
- void [configure](#) ([MQTT::Configuration](#) configuration)  
*Register new configuration.*

#### 8.17.1 Detailed Description

This class abstracts the process of publishing [MQTT](#) messages to an [MQTT](#) broker. Currently, the underlying implementation is realized using Paho-MQTT. Upon connection, it spans an own worker thread that continuously publishes messages put into a message queue such that the calling thread does not get blocked with the communication process. The purpose of this class is pure publishing, so there is no subscribing functionality implemented.

**Todo** Implement support for authenticating with client certificates.

Definition at line 35 of file [Publisher.h](#).

## 8.17.2 Constructor & Destructor Documentation

### 8.17.2.1 Publisher()

```
MQTT::Publisher::Publisher (
    std::string id,
    unsigned int buffer = 1024 )
```

Constructor for [MQTT](#) publisher that should be called from user's code

#### Parameters

in	<i>id</i>	Unique identifier that is presented to the broker. Be careful if you reuse code across applications that some broker show an unexpected behaviour when using the same id more than once.
in	<i>buffer</i>	Size of the message queue before messages with QoS=0 are discarded.

Definition at line 66 of file [Publisher.cpp](#).

### 8.17.2.2 ~Publisher()

```
MQTT::Publisher::~~Publisher ( )
```

Default destructor

Definition at line 82 of file [Publisher.cpp](#).

## 8.17.3 Member Function Documentation

### 8.17.3.1 configure()

```
void MQTT::Publisher::configure (
    MQTT::Configuration configuration )
```

Register a new configuration, without connecting. Changes will only take place after (re-)connecting to the broker.

#### Parameters

in	<i>configuration</i>	New configuration object.
----	----------------------	---------------------------

Definition at line 196 of file [Publisher.cpp](#).



### 8.17.3.2 connect() [1/2]

```
void MQTT::Publisher::connect ( )
```

Connect to the broker using the configuration object internally stored.

Definition at line 132 of file [Publisher.cpp](#).

### 8.17.3.3 connect() [2/2]

```
void MQTT::Publisher::connect (
    MQTT::Configuration configuration )
```

Connect to the broker using the provided configuration object. The internal configuration object will be overridden.

#### Parameters

in	<i>configuration</i>	<a href="#">Configuration</a> to apply.
----	----------------------	---

Definition at line 93 of file [Publisher.cpp](#).

### 8.17.3.4 disconnect()

```
void MQTT::Publisher::disconnect (
    unsigned int timeout = 10000 )
```

Disconnect from the broker and internally stop the worker thread. This is a blocking call.

#### Parameters

in	<i>timeout</i>	Timeout for the blocking operation in milliseconds.
----	----------------	---

Definition at line 157 of file [Publisher.cpp](#).

### 8.17.3.5 is\_connected()

```
bool MQTT::Publisher::is_connected (
    void )
```

Checks whether the publisher is currently connected to a broker. Directly calls the underlying implementation of the library if possible.

**Returns**

True if connected, else false.

Definition at line 184 of file [Publisher.cpp](#).

**8.17.3.6 min\_delay\_ms()**

```
int MQTT::Publisher::min_delay_ms (
    void ) [inline]
```

A minimum delay which should be kept between sending two messages, provided in milliseconds. Defaults to 0 .

**Postcondition**

This is not enforced in the client, but should be called as property of the publishing by the using code.

**Returns**

Current delay set.

Definition at line 261 of file [Publisher.h](#).

**8.17.3.7 publish() [1/2]**

```
bool MQTT::Publisher::publish (
    std::string topic,
    std::string message,
    int qos,
    bool retain )
```

This is the core function to publish an [MQTT](#) message. It deposits a message into the queue.

**Parameters**

in	<i>topic</i>	Topic to which the message shall be sent.
in	<i>message</i>	Primary content of the message to send.
in	<i>qos</i>	<a href="#">MQTT</a> Quality of service level to choose for the message. Check the <a href="#">MQTT</a> specifications for the exact behaviours of 0, 1, and 2 in conjunction with message retention.
in	<i>retain</i>	Flag whether to retain the message on the broker after disconnection.

**Returns**

True or false depending whether the message was accepted in the message queue (cf. buffer size).

Definition at line 207 of file [Publisher.cpp](#).

**8.17.3.8 publish()** [2/2]

```
bool MQTT::Publisher::publish (
    std::vector< std::string > topics,
    std::vector< std::string > messages,
    int qos,
    bool retain )
```

This is the multi-message version of the publish function, which deposits all messages at once into the message queue and gains performance by reducing the number of lock operations.

**Parameters**

in	<i>topic</i>	List of topics to which the messages shall be sent.
in	<i>message</i>	List of messages, in the corresponding order of the list of topics.
in	<i>qos</i>	<a href="#">MQTT</a> Quality of service level to choose for the message. Check the <a href="#">MQTT</a> specifications for the exact behaviours of 0, 1, and 2 in conjunction with message retention.
in	<i>retain</i>	Flag whether to retain the message on the broker after disconnection.

**Returns**

True or false depending whether the message was accepted in the message queue (cf. buffer size).

Definition at line [224](#) of file [Publisher.cpp](#).

**8.17.3.9 reconnect()**

```
void MQTT::Publisher::reconnect (
    void )
```

Direct call of the internal reconnect function. No reconfiguration takes place,

Definition at line [138](#) of file [Publisher.cpp](#).

**8.17.3.10 set\_buffer()**

```
void MQTT::Publisher::set_buffer (
    unsigned int buffer )
```

Set a new size for the internal message queue.

**Parameters**

in	<i>buffer</i>	New size
----	---------------	----------

Definition at line [179](#) of file [Publisher.cpp](#).

### 8.17.3.11 set\_root\_topic()

```
void MQTT::Publisher::set_root_topic (
    std::string root_topic )
```

Override the root topic. This can be done while connected as it only affects preprocessing of messages.

#### Parameters

in	<i>root_topic</i>	New root topic.
----	-------------------	-----------------

Definition at line 169 of file [Publisher.cpp](#).

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

- [src/MQTT/Publisher.h](#)
- [src/MQTT/Publisher.cpp](#)

## 8.18 SOIL::Range< T > Class Template Reference

[Range](#) Helper Class.

```
#include <Range.h>
```

### Public Member Functions

- [Range](#) ()  
*Default Constructor.*
- [Range](#) (T low, T high)  
*Argument constructor.*
- [Range](#) (std::vector< T > limits)  
*List constructor.*
- [~Range](#) ()  
*Destructor.*
- [HTTP::Json wjson](#) (void)  
*HTTP JSON.*
- bool [check](#) (const T &value)  
*Check.*

### 8.18.1 Detailed Description

```
template<typename T>
class SOIL::Range< T >
```

Class to represent the acceptable range of a figure. This class is templated to be applicable to all types. For numerical values, the range indicates a lower and upper limit which may be validly assigned. The limit values are included. The same holds for times. For strings, it indicates the minimum and maximum length. For enumerations, this is a list of applicable choices. For these types, the template is explicitly implemented differently. The range can be explicitly unset to provide an easy default member implementation.

There is no set function, if the range changes, a new instance of this lightweight class should be created.

## Template Parameters

<i>T</i>	Datatype for which the range is defined
----------	---

Definition at line 24 of file [Range.h](#).

## 8.18.2 Constructor & Destructor Documentation

### 8.18.2.1 Range() [1/3]

```
template<typename T >
SOIL::Range< T >::Range
```

Default Constructor, sets no limits and initializes `set` to false.

Definition at line 58 of file [Range.cpp](#).

### 8.18.2.2 Range() [2/3]

```
template<typename T >
SOIL::Range< T >::Range (
    T low,
    T high )
```

Constructs the range and initialized lower and upper limit with the provided values.

## Parameters

in	<i>low</i>	Lower limit
in	<i>high</i>	Upper limit

Definition at line 151 of file [Range.h](#).

### 8.18.2.3 Range() [3/3]

```
template<typename T >
SOIL::Range< T >::Range (
    std::vector< T > limits )
```

Constructs the range and initialized lower and upper limit with the provided vector. This is constructor can be conveniently initialized with the list notation {low, high}.

**Parameters**

<i>in</i>	<i>limits</i>	Vector of length 2 for initialization
-----------	---------------	---------------------------------------

Definition at line 156 of file [Range.h](#).

**8.18.2.4 ~Range()**

```
template<typename T >
SOIL::Range< T >::~~Range
```

Standard desctructor, no custom implementation.

Definition at line 68 of file [Range.cpp](#).

**8.18.3 Member Function Documentation****8.18.3.1 check()**

```
template<typename T >
bool SOIL::Range< T >::check (
    const T & value )
```

Check whether a given value is inside the prescribed range or not. If `set` is false, it always returns to true. The limits are included in the valid range, i.e. it is checked with `<=` and `>=`.

**Parameters**

<i>in</i>	<i>value</i>	Value to check
-----------	--------------	----------------

**Returns**

Check result als boolean, i.e. true if in range

Definition at line 85 of file [Range.cpp](#).

**8.18.3.2 wjson()**

```
template<typename T >
HTTP::Json SOIL::Range< T >::wjson (
    void )
```

Returns the [SOIL](#) conformant partial representation of the range of [Figure](#), i.e. `null` if not set.

### Returns

JSON [Object](#)

Definition at line 72 of file [Range.cpp](#).

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

- [src/SOIL/Range.h](#)
- [src/SOIL/Range.cpp](#)

## 8.19 SOIL::Range< ENUM > Class Reference

[Enum Range](#).

```
#include <Range.h>
```

### Public Member Functions

- [Range](#) ()
- [Range](#) (std::vector< std::string > choices)
- [~Range](#) ()
- [HTTP::Json wjson](#) (void)
- bool [check](#) (const std::string &value)
- bool [check](#) (const [SOIL::ENUM](#) &value)

### 8.19.1 Detailed Description

Specialization of [Range](#) for enumerations, where the limit is a set of applicable choices.

Definition at line 131 of file [Range.h](#).

### 8.19.2 Constructor & Destructor Documentation

#### 8.19.2.1 Range() [1/2]

```
SOIL::Range< ENUM >::Range ( )
```

#### 8.19.2.2 Range() [2/2]

```
SOIL::Range< ENUM >::Range (
    std::vector< std::string > choices )
```

### 8.19.2.3 ~Range()

```
SOIL::Range< ENUM >::~~Range ( )
```

## 8.19.3 Member Function Documentation

### 8.19.3.1 check() [1/2]

```
bool SOIL::Range< ENUM >::check (
    const SOIL::ENUM & value )
```

### 8.19.3.2 check() [2/2]

```
bool SOIL::Range< ENUM >::check (
    const std::string & value )
```

### 8.19.3.3 wjson()

```
HTTP::Json SOIL::Range< ENUM >::wjson (
    void )
```

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

- [src/SOIL/Range.h](#)

## 8.20 SOIL::Range< std::string > Class Reference

String [Range](#).

```
#include <Range.h>
```

### Public Member Functions

- [Range](#) ()
- [Range](#) (size\_t low, size\_t high)
- [Range](#) (std::vector< size\_t > limits)
- [~Range](#) ()
- [HTTP::Json wjson](#) (void)
- bool [check](#) (const std::string &value)



### 8.20.1 Detailed Description

Specialization of [Range](#) for strings, where the limits correspond to minimal and maximum length.

Definition at line 110 of file [Range.h](#).

### 8.20.2 Constructor & Destructor Documentation

#### 8.20.2.1 Range() [1/3]

```
SOIL::Range< std::string >::Range ( )
```

Definition at line 3 of file [Range.cpp](#).

#### 8.20.2.2 Range() [2/3]

```
SOIL::Range< std::string >::Range (
    size_t low,
    size_t high )
```

Definition at line 7 of file [Range.cpp](#).

#### 8.20.2.3 Range() [3/3]

```
SOIL::Range< std::string >::Range (
    std::vector< size_t > limits )
```

Definition at line 11 of file [Range.cpp](#).

#### 8.20.2.4 ~Range()

```
SOIL::Range< std::string >::~~Range ( )
```

Definition at line 29 of file [Range.cpp](#).

### 8.20.3 Member Function Documentation

### 8.20.3.1 check()

```
bool SOIL::Range< std::string >::check (
    const std::string & value )
```

Definition at line 44 of file [Range.cpp](#).

### 8.20.3.2 wjson()

```
web::json::value SOIL::Range< std::string >::wjson (
    void )
```

Definition at line 33 of file [Range.cpp](#).

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

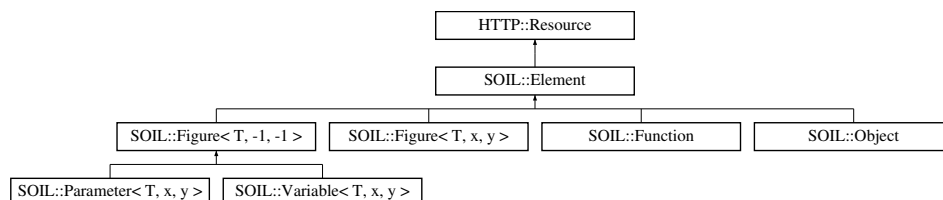
- [src/SOIL/Range.h](#)
- [src/SOIL/Range.cpp](#)

## 8.21 HTTP::Resource Class Reference

[HTTP Resource](#) base class.

```
#include <Resource.h>
```

Inheritance diagram for HTTP::Resource:



### Public Member Functions

- [Resource](#) ()  
*Constructor.*
- [~Resource](#) ()  
*Default Destructor.*
- virtual [Response handle](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP Handler.*
- virtual [Response handle\\_get](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP GET Handler.*
- virtual [Response handle\\_put](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP PUT Handler.*
- virtual [Response handle\\_post](#) ([Request](#) message, std::smatch match=std::smatch())

- virtual [Response handle\\_delete](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP POST Handler.*
- virtual [Response handle\\_patch](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP DELETE Handler.*
- virtual [Response handle\\_options](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP PATCH Handler.*
- virtual [Response handle\\_head](#) ([Request](#) message, std::smatch match=std::smatch())  
*HTTP OPTIONS Handler.*
- virtual [Response handle\\_exception](#) ([Request](#) message, std::exception &exception, std::smatch match=std::smatch())  
*HTTP HEAD Handler.*
- virtual [Response handle\\_exception](#) ([Request](#) message, std::exception &exception, std::smatch match=std::smatch())  
*HTTP Exception handler.*

## Static Public Member Functions

- static web::json::value [request\\_info](#) ([Request](#) message, std::smatch match=std::smatch())  
*Request Info.*

## Protected Member Functions

- void [apply\\_headers](#) ([Response](#) &response)  
*Apply headers.*

## Protected Attributes

- std::vector< web::http::method > [allowed\\_methods](#)  
*Allowed methods.*
- std::string [content\\_type](#)  
*Content type.*
- std::string [allowed\\_origins](#)  
*Allowed Origins.*

### 8.21.1 Detailed Description

Base class for any resource that is implemented using this library. It should always be subclassed by the class implementing the resource itself, and the relevant functions should be overridden. All [HTTP](#) business logic internally relies on the *cpprestsdk* library. The former internally uses *widestings*, which leads to the need for conversion work at some places.

Definition at line 20 of file [Resource.h](#).

### 8.21.2 Constructor & Destructor Documentation

### 8.21.2.1 Resource()

```
HTTP::Resource::Resource ( )
```

Default Constructor, sets the above mentioned defaults for `allowed_methods`, `content_type` and `allowed_origins`

Definition at line 67 of file [Resource.cpp](#).

### 8.21.2.2 ~Resource()

```
HTTP::Resource::~~Resource ( )
```

Default destructor, does no custom business logic.

Definition at line 74 of file [Resource.cpp](#).

## 8.21.3 Member Function Documentation

### 8.21.3.1 apply\_headers()

```
void HTTP::Resource::apply_headers (
    Response & response ) [protected]
```

Apply headers to the [HTTP](#) response object. This function should be called in the handler before returning the response object to the server.

#### Parameters

<code>in, out</code>	<code>response</code>	Response object to act on.
----------------------	-----------------------	----------------------------

Definition at line 50 of file [Resource.cpp](#).

### 8.21.3.2 handle()

```
HTTP::Response HTTP::Resource::handle (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests. By default, this function redirects to the more specific ones based on the respective [HTTP](#) Verb. In case an exception occurs, it calls the exception handler.

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented in [SOIL::Element](#).

Definition at line 78 of file [Resource.cpp](#).

### 8.21.3.3 `handle_delete()`

```
HTTP::Response HTTP::Resource::handle_delete (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a DELETE method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented in [SOIL::Object](#).

Definition at line 136 of file [Resource.cpp](#).

### 8.21.3.4 `handle_exception()`

```
HTTP::Response HTTP::Resource::handle_exception (
    Request message,
    std::exception & exception,
    std::smatch match = std::smatch() ) [virtual]
```

Handler function which can be called inside other handler functions to manage exceptions. It returns the request's information and exception error text while setting the [HTTP](#) response to 500 (Internal Error).

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>exception</i>	Exception to handle and to copy the error message from.
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cpprestsdk

Definition at line 158 of file [Resource.cpp](#).

**8.21.3.5 handle\_get()**

```
HTTP::Response HTTP::Resource::handle_get (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a GET method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

## Returns

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented in [SOIL::Object](#), [SOIL::Parameter< T, x, y >](#), [SOIL::Variable< T, x, y >](#), and [SOIL::Function](#).

Definition at line 120 of file [Resource.cpp](#).

**8.21.3.6 handle\_head()**

```
HTTP::Response HTTP::Resource::handle_head (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a HEAD method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

## Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Definition at line 153 of file [Resource.cpp](#).

**8.21.3.7 handle\_options()**

```
HTTP::Response HTTP::Resource::handle_options (  
    Request message,  
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on an OPTIONS method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented in [SOIL::Variable< T, x, y >](#).

Definition at line 148 of file [Resource.cpp](#).

**8.21.3.8 handle\_patch()**

```
HTTP::Response HTTP::Resource::handle_patch (  
    Request message,  
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a PATCH method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented in [SOIL::Parameter< T, x, y >](#).

Definition at line 130 of file [Resource.cpp](#).

### 8.21.3.9 `handle_post()`

```
HTTP::Response HTTP::Resource::handle_post (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a POST method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

#### Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cprestdsk
in	<i>match</i>	Match result of the request path that led to this resource

#### Returns

Outgoing [HTTP](#) response to be processed by cprestdsk

Reimplemented in [SOIL::Function](#).

Definition at line 125 of file [Resource.cpp](#).

### 8.21.3.10 `handle_put()`

```
HTTP::Response HTTP::Resource::handle_put (
    Request message,
    std::smatch match = std::smatch() ) [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a PUT method. This function should be overridden by the implementation of the resource. Per default, it resorts to the internal fallback function.

#### Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cprestdsk
in	<i>match</i>	Match result of the request path that led to this resource

#### Returns

Outgoing [HTTP](#) response to be processed by cprestdsk

Reimplemented in [SOIL::Object](#).

Definition at line 142 of file [Resource.cpp](#).



### 8.21.3.11 request\_info()

```
web::json::value HTTP::Resource::request_info (
    Request message,
    std::smatch match = std::smatch() ) [static]
```

Extracts all sort of information from the request and returns it in JSON format. This is useful for fallback methods.

#### Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cprestdk
in	<i>match</i>	Match result of the request path that led to this resource

#### Returns

Request information in JSON format

Definition at line 3 of file [Resource.cpp](#).

## 8.21.4 Member Data Documentation

### 8.21.4.1 allowed\_methods

```
std::vector<web::http::method> HTTP::Resource::allowed_methods [protected]
```

List of allowed [HTTP](#) methods for this resource implemented as `std::string` Defaults to all methods (GET, DELETE, PATCH, POST, OPTIONS, HEAD, PUT)

Definition at line 42 of file [Resource.h](#).

### 8.21.4.2 allowed\_origins

```
std::string HTTP::Resource::allowed_origins [protected]
```

Origins that are allowed, relevant when implementing web clients and running into CORS issues. Defaults to \* .

#### Postcondition

If you want to be serious about security, you should limit the origins here.

Definition at line 60 of file [Resource.h](#).

### 8.21.4.3 content\_type

```
std::string HTTP::Resource::content_type [protected]
```

Content type that is delivered by this resource. Defaults to `application/json`

Definition at line 50 of file [Resource.h](#).

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

- [src/REST/Resource.h](#)
- [src/REST/Resource.cpp](#)

## 8.22 HTTP::Server Class Reference

HTTP [Server](#).

```
#include <Server.h>
```

### Public Member Functions

- [Server](#) (std::string url)  
*Constructor.*
- [~Server](#) ()  
*Destructor.*
- void [open](#) ()  
*Start listener.*
- void [close](#) ()  
*Stop listener.*
- void [add](#) (std::string path, std::shared\_ptr< [Resource](#) > resource)  
*Register resource.*
- void [remove](#) (std::string path)  
*Register resource.*

### 8.22.1 Detailed Description

Abstraction layer for an [HTTP Server](#) implemented using the `cpprestsdk`.

**Todo** `cpprestsdk` is now in maintenance mode, if this library shall be in the very long term, a substitute may need to be found.

Definition at line 21 of file [Server.h](#).

### 8.22.2 Constructor & Destructor Documentation

#### 8.22.2.1 Server()

```
HTTP::Server::Server (  
    std::string url )
```

Constructor setting up the server, but not accepting connections upon construction.

## Parameters

in	<i>url</i>	Base URL, e.g. <code>http://localhost:8000</code> . Please note that the hostname part indicates the interface on which the server will listen.
----	------------	---

## Postcondition

When using another interface than localhost, Windows will prompt for firewall permissions. In general firewalls should be considered when working with [HTTP](#) Servers.

HTTPS is currently not implemented here - The current recommendation is that a reverse proxy is used for this purpose, e.g. NGINX or Microsoft IIS.

Definition at line 9 of file [Server.cpp](#).

## 8.22.2.2 ~Server()

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

Standard Destructor without custom logic. The server will be closed upon destruction, so make sure that the lifecycle of the server objects is appropriately managed in your application.

Definition at line 15 of file [Server.cpp](#).

## 8.22.3 Member Function Documentation

## 8.22.3.1 add()

```
void HTTP::Server::add (
    std::string path,
    std::shared_ptr< Resource > resource )
```

Register a resource by means of a matching expression and shared pointer reference. Please note that the order of adding resources to the sever determines the precedence in matching. The frst matching resource will handle the request. This is a common source of errors. This function can be called dynamically at runtime, also when the server is already running.

## Parameters

in	<i>path</i>	Path of the resource written as regular expression. Use <code>L"/?(.*)"</code> to match everything under the root path. The path simulatenously acts as unique identifier.
in	<i>resource</i>	Shared pointer to the resource that shall handle this path. Please be aware of the lifecycle in case you interfere with the pointers.

Definition at line 39 of file [Server.cpp](#).

### 8.22.3.2 close()

```
void HTTP::Server::close ( ) [inline]
```

Open the server for incoming connections. This is a blocking call.

Definition at line 91 of file [Server.h](#).

### 8.22.3.3 open()

```
void HTTP::Server::open ( ) [inline]
```

Open the server for incoming connections. This is a blocking call.

Definition at line 84 of file [Server.h](#).

### 8.22.3.4 remove()

```
void HTTP::Server::remove (
    std::string path )
```

Remove a previously registered resource. This function can be called dynamically at runtime, also when the server is already running.

#### Parameters

in	<i>path</i>	
		Path of the resource written as regular expression. Use <code>L"/?(.*)"</code> which was used to add the resource and acts an unique identifier.

Definition at line 44 of file [Server.cpp](#).

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

- [src/REST/Server.h](#)
- [src/REST/Server.cpp](#)

## 8.23 SIGN::Signer Class Reference

[Signer](#).

```
#include <Signer.h>
```

## Public Member Functions

- [Signer](#) (std::string filename)  
*Constructor.*
- [~Signer](#) ()  
*Destructor.*
- std::vector< unsigned char > [sign](#) (std::vector< unsigned char > digest)  
*Sign bytes.*
- std::string [openssl\\_version](#) (void) const  
*OpenSSL Version.*
- std::string [name](#) () const  
*Filename.*

### 8.23.1 Detailed Description

Class to provide a convenient interface to sign bytestrings using a private key. Internally based on OpenSSL.

Definition at line 18 of file [Signer.h](#).

### 8.23.2 Constructor & Destructor Documentation

#### 8.23.2.1 Signer()

```
SIGN::Signer::Signer (
    std::string filename )
```

Default constructor loading the private key in PEM-format and initializing necessary methods.

##### Parameters

in	<i>filename</i>	Path to the private key in PEM-format.
----	-----------------	--

Definition at line 11 of file [Signer.cpp](#).

#### 8.23.2.2 ~Signer()

```
SIGN::Signer::~~Signer ( )
```

Destructor, internally calls EVP\_PKEY\_free.

Definition at line 36 of file [Signer.cpp](#).

### 8.23.3 Member Function Documentation

#### 8.23.3.1 name()

```
std::string SIGN::Signer::name ( ) const [inline]
```

Function to retrieve the filename of the private key.

##### Returns

Filename of the private key.

Definition at line 86 of file [Signer.h](#).

#### 8.23.3.2 openssl\_version()

```
std::string SIGN::Signer::openssl_version (
    void ) const
```

Retreive the OpenSSL Version string. This is helpful to check whether the correct OpenSSL library has been loaded as by experience on windows there are many OpenSSL versions. Using a too old version of OpenSSL leads to very bad performance.

##### Returns

OpenSSL Version String

Definition at line 61 of file [Signer.cpp](#).

#### 8.23.3.3 sign()

```
std::vector< unsigned char > SIGN::Signer::sign (
    std::vector< unsigned char > digest )
```

Sign the bytestring provided to this function and returns the signature bytestring.

##### Precondition

It is assumed that the actual conteht to be signed has already been hashed, e.g. using SHA256 and the [Hasher](#) class.

## Parameters

<code>in</code>	<code>digest</code>	Bytes to sign, typically a digest from a previous hashing function
-----------------	---------------------	--

## Returns

Signature as vector of bytes

Definition at line 41 of file [Signer.cpp](#).

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

- [src/SIGN/Signer.h](#)
- [src/SIGN/Signer.cpp](#)

## 8.24 SOIL::Time Class Reference

[SOIL Time](#).

```
#include <Time.h>
```

### Public Member Functions

- DLL `std::string` [rfc3339](#) (void) const  
*RFC3339 representation.*
- DLL [Time](#) ()  
*Uninitialized Constructor.*
- DLL [Time](#) (std::string value)  
*String Constructor.*
- DLL [Time](#) (boost::posix\_time::ptime value)  
*Posix Time Construtor.*
- DLL [~Time](#) ()  
*Destructor.*
- bool [is\\_null](#) (void) const  
*Is Null?*
- void [set\\_null](#) (bool \_null=true)  
*Set null.*
- DLL `std::vector< unsigned char >` [serialize](#) (void) const  
*Bytewise serialization.*

### Static Public Member Functions

- static DLL boost::posix\_time::ptime [utc\\_now](#) (void)  
*Current Time.*

## Friends

- DLL bool `operator>=` (const [Time](#) &t1, const [Time](#) &t2)  
*Friend Operator >=.*
- DLL bool `operator<=` (const [Time](#) &t1, const [Time](#) &t2)  
*Friend Operator <=.*

### 8.24.1 Detailed Description

Speial [Time](#) class for [SOIL](#) to implement the necessary functionality.

Definition at line 12 of file [Time.h](#).

### 8.24.2 Constructor & Destructor Documentation

#### 8.24.2.1 `Time()` [1/3]

```
SOIL::Time::Time ( )
```

Construct an uninitialized [Time](#) instance, where the `_null` flag is set to true.

Definition at line 27 of file [Time.cpp](#).

#### 8.24.2.2 `Time()` [2/3]

```
SOIL::Time::Time (
    std::string value )
```

Constructs a [Time](#) object from an RFC3339 string. If an empty string is passed, the `_null` flag is set to true.

The exceptions from the underlying boost implementation are passed through.

##### Parameters

<code>in</code>	<code>value</code>	RFC3339 string of the time to set.
-----------------	--------------------	------------------------------------

Definition at line 31 of file [Time.cpp](#).

#### 8.24.2.3 `Time()` [3/3]

```
SOIL::Time::Time (
    boost::posix_time::ptime value )
```



Constructs a [Time](#) object from a `boost::posix_time::ptime`. This is useful in conjunction with `now()`.

#### Parameters

<code>in</code>	<code>value</code>	<a href="#">Time</a> to set.
-----------------	--------------------	------------------------------

Definition at line 46 of file [Time.cpp](#).

#### 8.24.2.4 ~Time()

```
SOIL::Time::~~Time ( )
```

Default Destructor, there is no unexpected behaviour.

Definition at line 53 of file [Time.cpp](#).

### 8.24.3 Member Function Documentation

#### 8.24.3.1 is\_null()

```
bool SOIL::Time::is_null (
    void ) const [inline]
```

Returns whether the time is set to null.

#### Returns

Is Null?

[Function](#) that returns true if the current data is set to null and false else.

#### Returns

Null flag

Definition at line 92 of file [Time.h](#).

### 8.24.3.2 rfc3339()

```
std::string SOIL::Time::rfc3339 (
    void ) const
```

Return a RFC3339 conformant representation of the time, e.g. 2021-06-07T22:01:23.078162Z as string. This is the main time representation used in [SOIL](#).

#### Returns

RFC3339 string

Definition at line 6 of file [Time.cpp](#).

### 8.24.3.3 serialize()

```
std::vector< unsigned char > SOIL::Time::serialize (
    void ) const
```

Byte-wise serialization of the time for hashing purpose. The components are serialized in the following order:

- uint16\_t of the year
- uint8\_t of the month
- uint8\_t of the day
- uint8\_t of the hour
- uint8\_t of the minute
- unit8\_t of the seconds
- uint32\_t of the nanoseconds

#### Returns

Serialized Bytestring

Definition at line 62 of file [Time.cpp](#).

### 8.24.3.4 set\_null()

```
void SOIL::Time::set_null (
    bool _null = true ) [inline]
```

[Function](#) to set the null status of the data container

## Parameters

<code>in</code>	<code>_null</code>	Boolean state flag
-----------------	--------------------	--------------------

Definition at line 100 of file [Time.h](#).

### 8.24.3.5 utc\_now()

```
boost::posix_time::ptime SOIL::Time::utc_now (
    void ) [static]
```

Returns the current UTC time as `boost::posix_time::ptime` to allow for quick access in implementations using this library. Thanks to the corresponding constructor, this function can also be used in assignments.

## Returns

Current UTC time

Definition at line 57 of file [Time.cpp](#).

## 8.24.4 Friends And Related Function Documentation

### 8.24.4.1 operator<=

```
DLL bool operator<= (
    const Time & t1,
    const Time & t2 ) [friend]
```

Friend declaration of the `<=` operator, needed for correct implementation.

Lower or equal operation for two [Time](#) objects, which is needed for the correct implementation of the [Range](#) class.

Returns true if `t1 <= t2`, otherwise false. If one of the [Time](#) objects is set to null, the operation returns false.

## Parameters

<code>t1</code>	Left hand side time
<code>t2</code>	Right hand side time

## Returns

Comparison result as boolean

#### 8.24.4.2 operator>=

```

DLL bool operator>= (
    const Time & t1,
    const Time & t2 ) [friend]

```

Friend declaration of the >= operator, needed for correct implementation.

Greater or equal operation for two [Time](#) objects, which is needed for the correct implementation of the [Range](#) class.

Returns true if  $t1 \geq t2$ , otherwise false. If one of the [Time](#) objects is set to null, the operation returns false.

##### Parameters

<i>t1</i>	Left hand side time
<i>t2</i>	Right hand side time

##### Returns

Comparison result as boolean

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

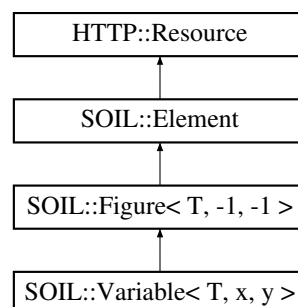
- [src/SOIL/Time.h](#)
- [src/SOIL/Time.cpp](#)

## 8.25 SOIL::Variable< T, x, y > Class Template Reference

[Variable](#) Class.

```
#include <Variable.h>
```

Inheritance diagram for SOIL::Variable< T, x, y >:



## Public Member Functions

- [Variable](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [unit](#), std::string [ontology](#)="", [Range](#)< T > [range](#)=[Range](#)< T >(), [TIME](#) [time](#)=[TIME](#)(), std::string [nonce](#)="")  
*Constructor.*
- [~Variable](#) ()  
*Destructor.*
- [Variable](#)< T, x, y > & [operator=](#) (const [Container](#)< T, x, y > &[value](#))  
*Assignment operator.*
- [HTTP::Json](#) [wjson](#) (void) override  
*HTTP JSON.*
- [HTTP::Response](#) [handle\\_get](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP GET Handler.*
- [HTTP::Response](#) [handle\\_options](#) ([HTTP::Request](#) message, std::smatch match=std::smatch()) override  
*HTTP OPTIONS Handler.*
- void [update](#) (const [Container](#)< T, x, y > &[value](#), [TIME](#) [time](#), std::string [nonce](#)="")  
*Update value.*
- void [set\\_covariance](#) ([Container](#)< T, x, x > [covariance](#))  
*Set Covariance.*
- std::vector< unsigned char > [bytes](#) (void)  
*Get bitwise representation.*
- std::vector< unsigned char > [sha256](#) (void)  
*Calculate SHA256.*
- std::vector< unsigned char > [sign](#) (std::shared\_ptr< [SIGN::Signer](#) > [signer](#)=NULL)  
*Sign the variable data.*
- std::vector< unsigned char > [fingerprint](#) (std::shared\_ptr< [SIGN::Signer](#) > [signer](#))  
*Calculate RSA fingerprint.*
- std::shared\_ptr< [Variable](#) > [ptr](#) (void)  
*Get Pointer.*
- bool [mqtt](#) (std::shared\_ptr< [MQTT::Publisher](#) > [publisher](#), int qos=0, bool retain=false)  
*Publish to MQTT.*

## Static Public Member Functions

- static std::shared\_ptr< [Variable](#) > [create](#) (std::shared\_ptr< [Element](#) > [parent](#), std::string [uuid](#), std::string [name](#), std::string [description](#), std::string [unit](#), std::string [ontology](#)="", [Range](#)< T > [range](#)=[Range](#)< T >(), [TIME](#) [time](#)=[TIME](#)(), std::string [nonce](#)="")  
*Create new Variable.*

## Protected Member Functions

- virtual void [read](#) (void)  
*Read callback.*
- virtual void [write](#) (void)  
*Write callback.*

## Protected Attributes

- `std::string` [nonce](#)  
*Nonce.*
- `std::vector< unsigned char >` [hash](#)  
*Checking Hash.*
- [Container](#)< T, x, x > [covariance](#)  
*Covariance of the value.*

## Additional Inherited Members

### 8.25.1 Detailed Description

```
template<typename T, int x = -1, int y = -1>
class SOIL::Variable< T, x, y >
```

This class represents a [SOIL Variable](#). [Parameter](#) and [Variable](#) share many common properties and therefore both inherit from [Figure](#), such that methods there should be considered in any case. Variables are primarily intended for values that represent any measurement or physical phenomenon outside the immediate control of device. In contrast to parameter, they cannot be set externally and may not be constant. In addition, they contain a custom message (nonce), a hash (for traceability purposes) and a covariance property. The covariance should represent uncertainty as multivariate normal distribution with coverage factor of 1.

Covariance is currently not supported for 2-dimensional data (i.e. which would lead to a 4-dimensional covariance expression)

In lightweight scenarios, this class may be instantiated directly, for more specific scenarios, it should be subclassed and override the implementation of `read()`.

A parameter supports [HTTP](#) GET (read) and [HTTP](#) OPTIONS (read without updating value) verbs.

The data management of [Figure](#) and hence [Parameter](#) relies on [Container](#), such that the same templating logic is used.

#### Template Parameters

<i>T</i>	Type of the data.
<i>x</i>	First dimension of the data. -1 means unused, 0 means arbitrary size. Cannot be -1 if <i>y</i> is not -1.
<i>y</i>	Second dimension of the data. -1 means unused, 0 means arbitrary size. Must be -1 for <i>x</i> to be -1.

**Todo** The [HTTP](#) handlers may be moved to protected if [HTTP::Server](#) is declared as friend class. Currently this is not done to allow for greater flexibility.

Definition at line 41 of file [Variable.h](#).

### 8.25.2 Constructor & Destructor Documentation

### 8.25.2.1 Variable()

```
template<typename T , int x, int y>
SOIL::Variable< T, x, y >::Variable (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    std::string ontology = "",
    Range< T > range = Range<T>(),
    TIME time = TIME(),
    std::string nonce = "" )
```

Standard constructor intialiazing the values. If subclassed, it should be called from the subclass constructor.

#### Parameters

in	<i>parent</i>	Shared pointer to parent object.
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed
in	<i>range</i>	Allowed range for the variable values, defaults to an empty object, i.e. allowing all values
in	<i>time</i>	Timestamp for the initial value, defaults to unset
in	<i>nonce</i>	Custom message to add to the variable, defaults to unset

Definition at line 312 of file [Variable.h](#).

### 8.25.2.2 ~Variable()

```
template<typename T , int x, int y>
SOIL::Variable< T, x, y >::~~Variable
```

Default destructor, without custom effort.

Definition at line 323 of file [Variable.h](#).

## 8.25.3 Member Function Documentation

### 8.25.3.1 bytes()

```
template<typename T , int x, int y>
std::vector< unsigned char > SOIL::Variable< T, x, y >::bytes (
    void ) [inline]
```

Get a bitwise representation of the variable. This is constructed in the following order:

- Dimensions in declared order
- Value in row-major order
- Covariance in row-major-order (cannot be null!)
- Unit directly taken from UTF-8 chars
- Timestamp (cf. explanation there)
- Nonce as UTF-8 string

This function is mainly used for hashing and signing purposes. It does not write to an internal attribute.

#### Returns

Constructucted bytestring

Definition at line 387 of file [Variable.h](#).

### 8.25.3.2 create()

```
template<typename T , int x, int y>
std::shared_ptr< SOIL::Variable< T, x, y > > SOIL::Variable< T, x, y >::create (
    std::shared_ptr< Element > parent,
    std::string uuid,
    std::string name,
    std::string description,
    std::string unit,
    std::string ontology = "",
    Range< T > range = Range<T>(),
    TIME time = TIME(),
    std::string nonce = "" ) [inline], [static]
```

Create a new [Variable](#) using the default constructor and return a shared pointer reference. This is the preferred method for manual creation with consistent lifecycle handling.

#### Parameters

in	<i>parent</i>	Shared pointer to parent object.
in	<i>uuid</i>	Locally Unique identifier
in	<i>name</i>	Human-readable name
in	<i>description</i>	Human-readable description
in	<i>unit</i>	UNECE unit code, e.g. MTR
in	<i>ontology</i>	Ontology reference, is set to null if an empty string is passed
in	<i>range</i>	Allowed range for the variable values, defaults to an empty object, i.e. allowing all values
in	<i>time</i>	Timestamp for the initial value, defaults to unset
in	<i>nonce</i>	Custom message to add to the variable, defaults to unset



Definition at line 328 of file [Variable.h](#).

### 8.25.3.3 fingerprint()

```
template<typename T , int x, int y>
std::vector< unsigned char > SOIL::Variable< T, x, y >::fingerprint (
    std::shared_ptr< SIGN::Signer > signer ) [inline]
```

Signs the sha256 hash with the private key managed by the passed instance of [SIGN::Signer](#). Does not write to an internal attribute.

#### Precondition

The [SIGN::Signer](#) instance must be instialized elsewhere and have an appropriate lifecycle.

#### Parameters

in	<i>signer</i>	Reference to a <a href="#">SIGN::Signer</a> object holding the private key to use
----	---------------	---

#### Returns

Signature bytestring

Definition at line 424 of file [Variable.h](#).

### 8.25.3.4 handle\_get()

```
template<typename T , int x, int y>
HTTP::Response SOIL::Variable< T, x, y >::handle_get (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [inline], [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a GET method. This function returns a representation of the [Variable](#) and its current value to the requesting party. It should not be overridden directly in subclasses, instead the [read\(\)](#) function should be overridden. It is an expected behaviour that [read\(\)](#) calls an external method to update the value and may take some time to return. Please note that the default timeout in cpprestsdk is 120s, which is already very long. Consider an asynchronous update model if your devices takes longer to measure.

#### Parameters

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 452 of file [Variable.h](#).

**8.25.3.5 handle\_options()**

```
template<typename T , int x, int y>
HTTP::Response SOIL::Variable< T, x, y >::handle_options (
    HTTP::Request message,
    std::smatch match = std::smatch() ) [inline], [override], [virtual]
```

Handler that is called by the server on [HTTP](#) requests on a OPTIONS method. This function returns a representation of the [Variable](#) and its current stored value to the requesting party. In contrast to GET, the [read\(\)](#) function is not called such that there are no side effects if a requesting party just wants to obtain information about the variable's metadata in the first place.

**Parameters**

in	<i>message</i>	Incoming <a href="#">HTTP</a> request as preprocessed by cpprestsdk
in	<i>match</i>	Match result of the request path that led to this resource

**Returns**

Outgoing [HTTP](#) response to be processed by cpprestsdk

Reimplemented from [HTTP::Resource](#).

Definition at line 464 of file [Variable.h](#).

**8.25.3.6 mqtt()**

```
template<typename T , int x, int y>
bool SOIL::Variable< T, x, y >::mqtt (
    std::shared_ptr< MQTT::Publisher > publisher,
    int qos = 0,
    bool retain = false ) [inline]
```

Publish the current JSON representation under the FQID as topic using a given [MQTT](#) publisher. This function must be explicitly called from the user's code as otherwise the update cycle would depend on the publisher and the user will be left without control to call other methods before publishing.

A good pattern is to have a reference to an [MQTT](#) publisher in a subclassing implementation and implement a complete update cycle there.

**Precondition**

An [MQTT::Publisher](#) must be instantiated elsewhere and have a valid lifecycle.

## Parameters

in	<i>publisher</i>	Reference to the publisher to use
in	<i>qos</i>	Quality of service to choose for <a href="#">MQTT</a> message
in	<i>retain</i>	Flag whether to retain the message on the server.

## Returns

Boolean flag whether the message was accepted in the message queue.

**Todo** Currently the implementation of this function is redundant in [Variable](#) and [Parameter](#). It is deliberately not moved to [Figure](#) as different implementations may occur in the future, but would be a valid option.

Definition at line [444](#) of file [Variable.h](#).

## 8.25.3.7 operator=()

```
template<typename T , int x, int y>
SOIL::Variable< T, x, y > & SOIL::Variable< T, x, y >::operator= (
    const Container< T, x, y > & value ) [inline]
```

Assigns the value provided as container on the right hand side to the variable. Immediately resorts to the implementation in [Figure](#) internally.

## Exceptions

<i>std::range_error</i>	Throws an exception if the value to assign is outside the allowed range.
-------------------------	--

## Parameters

in	<i>value</i>	Value to assign
----	--------------	-----------------

## Returns

Reference to the current [Variable](#)

Definition at line [335](#) of file [Variable.h](#).

## 8.25.3.8 ptr()

```
template<typename T , int x = -1, int y = -1>
std::shared_ptr< Variable > SOIL::Variable< T, x, y >::ptr (
    void ) [inline]
```

Return a shared pointer casted to the [Parameter](#) type to element itself.

**Returns**

Casted pointer

Definition at line 277 of file [Variable.h](#).

**8.25.3.9 read()**

```
template<typename T , int x, int y>
void SOIL::Variable< T, x, y >::read (
    void ) [protected], [virtual]
```

Read callback that can be implemented by deriving classes to perform custom build logic on read actions, e.g. update the value from an external storage. Be careful when including long-running queries as they will block the HTTP call. Declared virtual to make sure the derived method is called first. Does nothing by default.

Implements [SOIL::Figure< T, -1, -1 >](#).

Definition at line 370 of file [Variable.h](#).

**8.25.3.10 set\_covariance()**

```
template<typename T , int x, int y>
void SOIL::Variable< T, x, y >::set_covariance (
    Container< T, x, x > covariance ) [inline]
```

Set the covariance corresponding to the value. This is a separate call as the following three scenarios are foreseen:

- The covariance is set in the user defined update cycle, so this call can directly follow [update\(\)](#).
- The covariance is estimated constant for all measurements, so there is no need for updating it
- The covariance is not available, thus set to null.

**Parameters**

in	<i>covariance</i>	Covariance provided as container object
----	-------------------	---

Definition at line 380 of file [Variable.h](#).

**8.25.3.11 sha256()**

```
template<typename T , int x, int y>
std::vector< unsigned char > SOIL::Variable< T, x, y >::sha256 (
    void ) [inline]
```

Calculates the sha256 hash of the bytestring representation. Internally calls `bytes()`, but does not write to any value.

#### Returns

SHA256 hash as bytestring

Definition at line 417 of file [Variable.h](#).

### 8.25.3.12 `sign()`

```
template<typename T , int x, int y>
std::vector< unsigned char > SOIL::Variable< T, x, y >::sign (
    std::shared_ptr< SIGN::Signer > signer = NULL ) [inline]
```

Sign the measurement data. If an pointer to an instance of [SIGN::Signer](#) is passed, the `fingerprint()` function is used to sign the sha256 hash with the managed private key. Otherwise, the `sha256()` function is used.

In both cases, the resulting bytestring is stored to the internal `hash` attribute and returned.

#### Precondition

The [SIGN::Signer](#) instance must be instialized elsewhere and have an aproprate lifecycle.

#### Postcondition

The `hash` attribute is automatically updated by this call.

#### Parameters

in	<i>signer</i>	Reference to a <a href="#">SIGN::Signer</a> object holding the private key to use
----	---------------	---

#### Returns

Signature bytestring

Definition at line 430 of file [Variable.h](#).

### 8.25.3.13 `update()`

```
template<typename T , int x, int y>
void SOIL::Variable< T, x, y >::update (
    const Container< T, x, y > & value,
    TIME time,
    std::string nonce = "" ) [inline]
```

Update the value of the variable. As value, time and nonce are expected to change with each value update, the three components are passed at the same time.

**Parameters**

in	<i>value</i>	Value to set
in	<i>time</i>	Timestamp to set as corresponding to the value
in	<i>nonce</i>	Nonce to set as corresponding to the value

Definition at line 474 of file [Variable.h](#).

**8.25.3.14 wjson()**

```
template<typename T , int x, int y>
HTTP::Json SOIL::Variable< T, x, y >::wjson (
    void ) [override], [virtual]
```

Get a [HTTP](#) JSON object corresponding to the current state of the [Parameter](#). This function provides a SOIL-conformant JSON representation the parameter. It internally extends the method of [Figure](#).

**Returns**

JSON object

Reimplemented from [SOIL::Element](#).

Definition at line 343 of file [Variable.h](#).

**8.25.3.15 write()**

```
template<typename T , int x, int y>
void SOIL::Variable< T, x, y >::write (
    void ) [protected], [virtual]
```

This is an empty implementation of the [write\(\)](#) function which is needed to avoid that [Variable](#) becomes abstract. It is never called.

Implements [SOIL::Figure< T, -1, -1 >](#).

Definition at line 375 of file [Variable.h](#).

**8.25.4 Member Data Documentation**

#### 8.25.4.1 covariance

```
template<typename T , int x = -1, int y = -1>
SOIL::Variable< T, x, y >::covariance [protected]
```

Covariance of the value expressed as multidimensional normal distribution with coverage factor 1. The dimensionality depends on the dimensionality of the value:

- For scalar data, the covariance is a scalar corresponding to the variance.
- For data expanding along one dimension with length  $n$  the covariance is a matrix with dimension  $(n,n)$
- For data expanding along two dimensions, covariance is currently not supported.

Note that the second template argument of the [Container](#) is  $x$  in this case.

Definition at line 74 of file [Variable.h](#).

#### 8.25.4.2 hash

```
template<typename T , int x = -1, int y = -1>
std::vector<unsigned char> SOIL::Variable< T, x, y >::hash [protected]
```

Hash that is calculated from a defined binary representation of the variable and may be signed with `private_key` such that the integrity of the data can be verified. Is null if not explicitly set.

**Todo** This is still error prone across different languages and platforms as endianness and memory layout need to be considered.

Definition at line 61 of file [Variable.h](#).

#### 8.25.4.3 nonce

```
template<typename T , int x = -1, int y = -1>
std::string SOIL::Variable< T, x, y >::nonce [protected]
```

Nonce with is considered as arbitrary message that the user may add to the measurement. This could be JOB-IDs or similar.

Definition at line 51 of file [Variable.h](#).

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

- [src/SOIL/Variable.h](#)





## Chapter 9

# File Documentation

### 9.1 README.md File Reference

### 9.2 src/MQTT/Configuration.cpp File Reference

```
#include "Configuration.h"  
#include <nlohmann/json.hpp>  
#include <fstream>
```

#### Typedefs

- using `json` = `nlohmann::json`

#### 9.2.1 Typedef Documentation

##### 9.2.1.1 json

```
using json = nlohmann::json
```

Definition at line 6 of file [Configuration.cpp](#).

## 9.3 Configuration.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Configuration.h"
00002 #include <nlohmann/json.hpp>
00003 #include <fstream>
00004
00005
00006 using json = nlohmann::json;
00007
00008 MQTT::Configuration::Configuration()
00009 {
00010     std::string host = "127.0.0.1";
00011     port = 1883;
00012     username = "guest";
00013     password = "guest";
00014     clean_session = true;
00015     root = "";
00016     keep_alive = 30;
00017     min_delay_ms = 0;
00018     int connection_timeout_s = 30;
00019     bool ssl = false;
00020     bool verify = true;
00021     bool websocket = false;
00022     std::string path = "";
00023     std::string certificate_authority = "";
00024     connection_timeout_s = 30;
00025 }
00026
00027 MQTT::Configuration::Configuration(std::string filename)
00028 {
00029     std::ifstream infile(filename);
00030     json j;
00031     infile >> j;
00032     host = j.value("host", "127.0.0.1");
00033     port = j.value("port", 1883);
00034     username = j.value("username", "guest");
00035     password = j.value("password", "guest");
00036     clean_session = j.value("clean", true);
00037     root = j.value("root", "");
00038     keep_alive = j.value("keep_alive", 30);
00039     min_delay_ms = j.value("min_delay", 0);
00040     connection_timeout_s = j.value("connection_timeout", 30);
00041     ssl = j.value("ssl", false);
00042     verify = j.value("verify", true);
00043     websocket = j.value("websocket", false);
00044     path = j.value("path", "");
00045     certificate_authority = j.value("certificate_authority", "");
00046 }
00047
00048
00049 MQTT::Configuration::~Configuration()
00050 {
00051 }
00052
00053 std::string MQTT::Configuration::uri()
00054 {
00055     std::string prefix;
00056     if (websocket)
00057     {
00058         if (ssl)
00059         {
00060             prefix = "wss";
00061         }
00062         else
00063         {
00064             prefix = "ws";
00065         }
00066         if (path.size() > 0)
00067         {
00068             if (path.at(0) != '/')
00069             {
00070                 path = "/" + path;
00071             }
00072         }
00073     }
00074     else
00075     {
00076         if (ssl)
00077         {
00078             prefix = "ssl";
00079         }
00080         else
00081         {
00082             prefix = "tcp";

```

```

00083         }
00084         path = "";
00085     }
00086
00087     return prefix + "://" + host + ":" + std::to_string(port) + path;
00088
00089 }
00090

```

## 9.4 src/UDP/Configuration.cpp File Reference

```

#include "Configuration.h"
#include <nlohmann/json.hpp>
#include <fstream>

```

### Typedefs

- using `json` = `nlohmann::json`

### 9.4.1 Typedef Documentation

#### 9.4.1.1 json

```
using json = nlohmann::json
```

Definition at line 5 of file [Configuration.cpp](#).

## 9.5 Configuration.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Configuration.h"
00002 #include <nlohmann/json.hpp>
00003 #include <fstream>
00004
00005 using json = nlohmann::json;
00006
00007 UDP::Configuration::Configuration()
00008 {
00009     clients.clear();
00010 }
00011
00012 UDP::Configuration::Configuration(std::string filename)
00013 {
00014     std::ifstream infile(filename);
00015     json j;
00016     infile » j;
00017
00018     for (json::iterator it = j["clients"].begin(); it != j["clients"].end(); it++)
00019     {
00020         clients[(*it)["address"]] = (*it)["port"];
00021     }
00022 }
00023
00024 UDP::Configuration::~~Configuration()
00025 {
00026     clients.clear();
00027 }

```

## 9.6 src/MQTT/Configuration.h File Reference

```
#include <string>
#include "constants.h"
```

### Classes

- class [MQTT::Configuration](#)  
*MQTT publishing configuration.*

### Namespaces

- namespace [MQTT](#)

## 9.7 Configuration.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <string>
00003 #include "constants.h"
00004
00005 namespace MQTT
00006 {
00013     class DLL Configuration
00014     {
00015     public:
00022         std::string host;
00023
00030         int port;
00031
00039         std::string username;
00040
00047         std::string password;
00048
00055         bool clean_session;
00056
00064         std::string root;
00065
00072         int keep_alive;
00073
00081         int min_delay_ms;
00082
00090         int connection_timeout_s;
00091
00099         bool ssl;
00100
00107         bool verify;
00108
00116         bool websocket;
00117
00124         std::string path;
00125
00133         std::string certificate_authority;
00134
00140         Configuration();
00141
00149         Configuration(std::string filename);
00150
00156         ~Configuration();
00157
00164         std::string uri();
00165     };
00166 }
```

## 9.8 src/UDP/Configuration.h File Reference

```
#include "constants.h"
#include <map>
#include <string>
```

### Classes

- class [UDP::Configuration](#)  
*UDP Broadcast Configuration.*

### Namespaces

- namespace [UDP](#)

## 9.9 Configuration.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include <map>
00004 #include <string>
00005 namespace UDP
00006 {
00013     class DLL Configuration
00014     {
00015     public:
00021         typedef std::map<std::string, int>::iterator iterator;
00022
00028         std::map<std::string, int> clients;
00029
00035         Configuration();
00036
00044         Configuration(std::string filename);
00045
00051         ~Configuration();
00052     };
00053 }
```

## 9.10 src/MQTT/constants.h File Reference

### Macros

- #define [\\_WIN32\\_WINNT](#) 0x600  
*Default definitions for project.*

### 9.10.1 Macro Definition Documentation

### 9.10.1.1 `_WIN32_WINNT`

```
#define _WIN32_WINNT 0x600
```

File being referenced by many others containing cross-project defaults.

Definition at line 20 of file [constants.h](#).

## 9.11 `constants.h`

[Go to the documentation of this file.](#)

```
00001
00007 #pragma once
00008 #ifdef _WIN32
00009 #     pragma warning(disable: 4251)
00010 #     pragma warning(disable: 4275)
00011 #     ifdef _WINDLL
00012 #         define DLL __declspec(dllexport)
00013 #     else
00014 #         define DLL __declspec(dllimport)
00015 #     endif
00016 #elif defined(__unix__)
00017 #     define DLL __attribute__((visibility("default")))
00018 #endif
00019
00020 #define _WIN32_WINNT 0x600
00021 #ifdef _WIN32
00022     #include <SDKDDKVer.h>
00023 #endif
```

## 9.12 `src/REST/constants.h` File Reference

### Macros

- `#define _WIN32_WINNT 0x600`

*Default definitions for project.*

### 9.12.1 Macro Definition Documentation

#### 9.12.1.1 `_WIN32_WINNT`

```
#define _WIN32_WINNT 0x600
```

File being referenced by many others containing cross-project defaults.

Definition at line 19 of file [constants.h](#).

## 9.13 constants.h

[Go to the documentation of this file.](#)

```
00001
00006 #pragma once
00007 #ifndef _WIN32
00008 #     pragma warning(disable: 4251)
00009 #     pragma warning(disable: 4275)
00010 #     ifdef _WINDLL
00011 #         define DLL __declspec(dllexport)
00012 #     else
00013 #         define DLL __declspec(dllimport)
00014 #     endif
00015 #elif defined(__unix__)
00016 #     define DLL __attribute__((visibility("default")))
00017 #endif
00018
00019 #define _WIN32_WINNT 0x600
00020 #ifndef _WIN32
00021 #include <SDKDDKVer.h>
00022 #endif
```

## 9.14 src/SIGN/constants.h File Reference

### Macros

- `#define _WIN32_WINNT 0x600`  
*Default definitions for project.*

### 9.14.1 Macro Definition Documentation

#### 9.14.1.1 \_WIN32\_WINNT

```
#define _WIN32_WINNT 0x600
```

File being referenced by many others containing cross-project defaults.

Definition at line 19 of file [constants.h](#).

## 9.15 constants.h

[Go to the documentation of this file.](#)

```
00001
00006 #pragma once
00007 #ifndef _WIN32
00008 #     pragma warning(disable: 4251)
00009 #     pragma warning(disable: 4275)
00010 #     ifdef _WINDLL
00011 #         define DLL __declspec(dllexport)
00012 #     else
00013 #         define DLL __declspec(dllimport)
00014 #     endif
00015 #elif defined(__unix__)
00016 #     define DLL __attribute__((visibility("default")))
00017 #endif
00018
00019 #define _WIN32_WINNT 0x600
00020 #ifndef _WIN32
00021 #include <SDKDDKVer.h>
00022 #endif
```

## 9.16 src/SOIL/constants.h File Reference

### Macros

- `#define _WIN32_WINNT 0x600`  
*Default definitions for project.*

### 9.16.1 Macro Definition Documentation

#### 9.16.1.1 \_WIN32\_WINNT

```
#define _WIN32_WINNT 0x600
```

File being referenced by many others containing cross-project defaults.

Definition at line 19 of file [constants.h](#).

## 9.17 constants.h

[Go to the documentation of this file.](#)

```
00001
00006 #pragma once
00007 #ifndef _WIN32
00008 #     pragma warning(disable: 4251)
00009 #     pragma warning(disable: 4275)
00010 #     ifdef _WINDLL
00011 #         define DLL __declspec(dllexport)
00012 #     else
00013 #         define DLL __declspec(dllimport)
00014 #     endif
00015 #elif defined(__unix__)
00016 #     define DLL __attribute__((visibility("default")))
00017 #endif
00018
00019 #define _WIN32_WINNT 0x600
00020 #ifndef _WIN32
00021 #include <SDKDDKVer.h>
00022 #endif
```

## 9.18 src/UDP/constants.h File Reference

### 9.19 constants.h

[Go to the documentation of this file.](#)

```
00001
00006 #pragma once
00007 #ifndef _WIN32
00008 #     pragma warning(disable: 4251)
00009 #     pragma warning(disable: 4275)
00010 #     ifdef _WINDLL
00011 #         define DLL __declspec(dllexport)
00012 #     else
00013 #         define DLL __declspec(dllimport)
00014 #     endif
00015 #elif defined(__unix__)
00016 #     define DLL __attribute__((visibility("default")))
00017 #endif
00018
00019 #ifndef _WIN32
00020 #include <SDKDDKVer.h>
00021 #endif
```



## 9.20 src/MQTT/LocalException.cpp File Reference

```
#include "LocalException.h"
```

### 9.21 LocalException.cpp

[Go to the documentation of this file.](#)

```
00001 #include "LocalException.h"
00002
00003
00004
00005 MQTT::Exception::Exception(const char* message, int code): std::runtime_error(message), _code(code)
00006 {
00007 }
00008
00009 MQTT::Exception::Exception(const std::exception & exc) : std::runtime_error(exc.what())
00010 {
00011     _code = 0;
00012 }
00013
00014 MQTT::Exception::~Exception()
00015 {
00016 }
00017
00018
```

## 9.22 src/MQTT/LocalException.h File Reference

```
#include "constants.h"
#include <stdexcept>
```

### Classes

- class [MQTT::Exception](#)  
*MQTT Publisher Exception.*

### Namespaces

- namespace [MQTT](#)

## 9.23 LocalException.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include <stdexcept>
00004
00005
00006 namespace MQTT
00007 {
00013     class DLL Exception : public std::runtime_error
00014     {
00015     private:
00016
```

```

00022         int _code;
00023     public:
00024
00032         Exception(const char* message = "", int code = 0);
00033
00040         Exception(const std::exception& exc);
00041
00047         ~Exception(void);
00048
00055         inline const int code(void) const { return _code; }
00056     };
00057 }
00058
00059
00060

```

## 9.24 src/MQTT/MessageContainer.h File Reference

```
#include "constants.h"
```

### Classes

- struct [MQTT::MessageContainer](#)  
Internal *MQTT* message struct.

### Namespaces

- namespace [MQTT](#)

## 9.25 MessageContainer.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 namespace MQTT
00004 {
00010     struct DLL MessageContainer
00011     {
00017         std::string topic;
00018
00024         std::string message;
00025
00032         int qos;
00033
00039         bool retain;
00040
00041     };
00042 }

```

## 9.26 src/MQTT/Publisher.cpp File Reference

```

#include "LocalException.h"
#include "Publisher.h"
#include <nlohmann/json.hpp>
#include <regex>
#include <chrono>
#include <condition_variable>
#include <mqtt/async_client.h>
#include <boost/date_time/posix_time/posix_time.hpp>

```

## Macros

- `#define PAHO_MQTT_IMPORTS`

### 9.26.1 Macro Definition Documentation

#### 9.26.1.1 PAHO\_MQTT\_IMPORTS

```
#define PAHO_MQTT_IMPORTS
```

Definition at line 7 of file [Publisher.cpp](#).

## 9.27 Publisher.cpp

[Go to the documentation of this file.](#)

```
00001 #include "LocalException.h"
00002 #include "Publisher.h"
00003 #include <nlohmann/json.hpp>
00004 #include <regex>
00005 #include <chrono>
00006 #include <condition_variable>
00007 #define PAHO_MQTT_IMPORTS
00008 #include <mqtt/async_client.h>
00009 #include <boost/date_time/posix_time/posix_time.hpp>
00010
00011
00012 int MQTT::Publisher::instances = 0;
00013
00014
00015
00016 void MQTT::Publisher::loop(void)
00017 {
00018     MQTT::MessageContainer message;
00019     while (worker_running.load(std::memory_order_consume))
00020     {
00021         if (queue_size.load(std::memory_order_consume) > 0)
00022         {
00023             std::unique_lock<std::mutex> queue_lock(queue_mutex);
00024             message = queue.front();
00025             queue.pop_front();
00026             queue_size.store(static_cast<int>(queue.size()));
00027         }
00028         else
00029         {
00030             std::unique_lock<std::mutex> queue_lock(queue_mutex);
00031             while (queue_size.load(std::memory_order_consume) == 0)
00032             {
00033                 queue_condition.wait(queue_lock);
00034             }
00035             continue;
00036         }
00037         try {
00038             auto paho_message = mqtt::make_message((configuration.root + message.topic).c_str(),
00039             message.message.c_str(), message.message.size(), message.qos, message.retain);
00040             auto token = client->publish(paho_message);
00041         } catch (std::exception&)
00042         {
00043             /* pass */
00044         }
00045     }
00046 }
00047
00048 void MQTT::Publisher::restart_worker(void)
00049 {
00050     if (!worker_running.load(std::memory_order_consume))
00051     {
00052         std::unique_lock<std::mutex> queue_lock(queue_mutex);
```

```

00053         queue.clear();
00054         worker_running.store(true, std::memory_order_release);
00055         worker.reset(new std::thread(&MQTT::Publisher::loop, this));
00056     }
00057 }
00058
00059 void MQTT::Publisher::stop_worker(void)
00060 {
00061     worker_running.store(false, std::memory_order_release);
00062     std::unique_lock<std::mutex> queue_lock(queue_mutex);
00063     queue_condition.notify_all();
00064 }
00065
00066 MQTT::Publisher::Publisher(std::string id, unsigned int buffer) : id(id), buffer(buffer)
00067 {
00068     if (instances == 0)
00069     {
00070         /* pass */;
00071     }
00072     instances++;
00073
00074     worker_running.store(false, std::memory_order_release);
00075     queue.resize(0);
00076     queue_size.store(0, std::memory_order_release);
00077 }
00078
00079
00080
00081
00082 MQTT::Publisher::~~Publisher()
00083 {
00084     this->disconnect();
00085     worker->detach();
00086     instances--;
00087     if (instances == 0)
00088     {
00089         /* pass */;
00090     }
00091 }
00092
00093 void MQTT::Publisher::connect(MQTT::Configuration configuration)
00094 {
00095     try
00096     {
00097         client.reset(new mqtt::async_client(configuration.uri(), id));
00098
00099         auto ssl = mqtt::ssl_options_builder();
00100         auto options = mqtt::connect_options_builder();
00101         options.user_name(configuration.username);
00102         options.password(configuration.password);
00103         options.clean_session(configuration.clean_session);
00104         options.keep_alive_interval(std::chrono::seconds(configuration.keep_alive));
00105         options.connect_timeout(std::chrono::seconds(configuration.connection_timeout_s));
00106
00107         if (configuration.ssl)
00108         {
00109             ssl.enable_server_cert_auth(configuration.verify);
00110             ssl.trust_store(configuration.certificate_authority);
00111             ssl.ssl_version(MQTT_SSL_VERSION_DEFAULT);
00112             options.ssl(ssl.finalize());
00113         }
00114
00115         auto token = client->connect(options.finalize());
00116         auto response = token->get_connect_response();
00117
00118         if (!client->is_connected())
00119         {
00120             throw std::runtime_error(mqtt::exception(token->get_return_code()));
00121         }
00122
00123         restart_worker();
00124     }
00125     catch (std::exception& exc)
00126     {
00127         throw MQTT::Exception(exc);
00128     }
00129 }
00130
00131
00132 void MQTT::Publisher::connect()
00133 {
00134     this->connect(this->configuration);
00135 }
00136
00137
00138 void MQTT::Publisher::reconnect(void)
00139 {

```

```

00140     try
00141     {
00142         auto token = client->reconnect();
00143         auto response = token->get_connect_response();
00144
00145         if (!client->is_connected())
00146         {
00147             throw std::runtime_error(mqtt::exception(token->get_return_code()));
00148         }
00149         restart_worker();
00150     }
00151     catch (std::exception& exc)
00152     {
00153         throw MQTT::Exception(exc);
00154     }
00155 }
00156
00157 void MQTT::Publisher::disconnect(unsigned int timeout)
00158 {
00159     if (client.get() != NULL)
00160     {
00161         auto token = client->disconnect();
00162         token->wait_for(timeout);
00163     }
00164     stop_worker();
00165     client.reset();
00166 }
00167
00168
00169 void MQTT::Publisher::set_root_topic(std::string root_topic)
00170 {
00171     configuration.root = root_topic;
00172     std::string::iterator last_character = configuration.root.end() - 1;
00173     if ((*last_character) != '/')
00174     {
00175         configuration.root += "/";
00176     }
00177 }
00178
00179 void MQTT::Publisher::set_buffer(unsigned int buffer)
00180 {
00181     this->buffer = buffer;
00182 }
00183
00184 bool MQTT::Publisher::is_connected(void)
00185 {
00186     try
00187     {
00188         return client->is_connected();
00189     }
00190     catch (std::exception& exc)
00191     {
00192         throw MQTT::Exception(exc);
00193     }
00194 }
00195
00196 void MQTT::Publisher::configure(MQTT::Configuration configuration)
00197 {
00198     this->configuration = configuration;
00199     std::string::iterator last_character = this->configuration.root.end() - 1;
00200     if ((*last_character) != '/')
00201     {
00202         this->configuration.root += "/";
00203     }
00204 }
00205
00206
00207 bool MQTT::Publisher::publish(std::string topic, std::string message, int qos, bool retain)
00208 {
00209     MQTT::MessageContainer mqtt_message = { topic, message, qos, retain };
00210     if ((qos > 0) || (queue_size.load(std::memory_order_consume) <= static_cast<int>(buffer)))
00211     {
00212         std::unique_lock<std::mutex> queue_lock(queue_mutex);
00213         queue.push_back(mqtt_message);
00214         queue_size.store(static_cast<int>(queue.size()), std::memory_order_release);
00215         queue_condition.notify_all();
00216         return true;
00217     }
00218     else
00219     {
00220         return false;
00221     }
00222 }
00223
00224 bool MQTT::Publisher::publish(std::vector<std::string> topics, std::vector<std::string> messages, int
qos, bool retain)
00225 {

```

```

00226     int n = static_cast<int>(topics.size());
00227     std::vector<MQTT::MessageContainer> mqtt_messages(n);
00228     for (int i = 0; i < n; i++)
00229     {
00230         mqtt_messages.at(i) = { topics.at(i), messages.at(i), qos, retain };
00231     }
00232
00233     if ((qos > 0) || (queue_size.load(std::memory_order_consume) <= static_cast<int>(buffer)))
00234     {
00235         std::unique_lock<std::mutex> queue_lock(queue_mutex);
00236         queue.insert(queue.end(), mqtt_messages.begin(), mqtt_messages.end());
00237         queue_size.store(static_cast<int>(queue.size()), std::memory_order_release);
00238         queue_condition.notify_all();
00239         return true;
00240     }
00241     else
00242     {
00243         return false;
00244     }
00245 }
00246 }

```

## 9.28 src/MQTT/Publisher.h File Reference

```

#include "constants.h"
#include "Configuration.h"
#include "MessageContainer.h"
#include <deque>
#include <atomic>
#include <mutex>
#include <thread>
#include <vector>
#include <condition_variable>

```

### Classes

- class [MQTT::Publisher](#)  
*MQTT Publisher.*

### Namespaces

- namespace [mqtt](#)
- namespace [MQTT](#)

## 9.29 Publisher.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "Configuration.h"
00004 #include "MessageContainer.h"
00005 #include <deque>
00006 #include <atomic>
00007 #include <mutex>
00008 #include <thread>
00009 #include <vector>
00010 #include <condition_variable>
00011 // #include <mqtt/async_client.h>
00012
00013

```

```

00014 namespace mqtt
00015 {
00019     class async_client;
00020 }
00021
00022 namespace MQTT
00023 {
00035     class DLL Publisher
00036     {
00037     private:
00044         static int instances;
00045
00052         std::string id;
00053
00062         unsigned int buffer;
00063
00069         Configuration configuration;
00070
00077         std::shared_ptr<mqtt::async_client> client;
00078
00084         std::deque<MessageContainer> queue;
00085
00091         std::atomic<int> queue_size;
00092
00098         std::mutex queue_mutex;
00099
00105         std::condition_variable queue_condition;
00106
00113         std::shared_ptr<std::thread> worker;
00114
00121         std::atomic<bool> worker_running;
00122
00130         void loop(void);
00131
00138         void restart_worker(void);
00139
00145         void stop_worker(void);
00146
00147
00148     public:
00157         Publisher(std::string id, unsigned int buffer = 1024);
00158
00164         ~Publisher();
00165
00177         bool publish(std::string topic, std::string message, int qos, bool retain);
00178
00190         bool publish(std::vector<std::string> topics, std::vector<std::string> messages, int qos, bool
retain);
00191
00200         void connect(MQTT::Configuration configuration);
00201
00207         void connect();
00208
00214         void reconnect(void);
00215
00224         void disconnect(unsigned int timeout = 10000);
00225
00232         void set_root_topic(std::string root_topic);
00233
00241         void set_buffer(unsigned int buffer);
00242
00251         bool is_connected(void);
00252
00261         inline int min_delay_ms(void) { return configuration.min_delay_ms; }
00262
00270         void configure(MQTT::Configuration configuration);
00271
00272
00273     };
00274 }
00275
00276

```

## 9.30 src/REST/Resource.cpp File Reference

```
#include "Resource.h"
```

## 9.31 Resource.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Resource.h"
00002
00003 web::json::value HTTP::Resource::request_info(Request message, std::smatch match)
00004 {
00005     web::json::value headers;
00006     for (web::http::http_headers::iterator i = message.headers().begin(); i !=
00007         message.headers().end(); i++)
00008     {
00009         headers[i->first] = web::json::value::string(i->second);
00010     }
00011     web::json::value body;
00012     body[U("method")] = web::json::value::string(message.method());
00013     body[U("headers")] = headers;
00014     auto task = message.extract_string();
00015     task.wait();
00016     body[U("body")] = web::json::value::string(task.get());
00017     web::uri uri = message.absolute_uri();
00018     web::json::value url;
00019     url[U("path")] = web::json::value::string(uri.path());
00020     url[U("scheme")] = web::json::value::string(uri.scheme());
00021     url[U("query")] = web::json::value::string(uri.query());
00022     url[U("host")] = web::json::value::string(uri.host());
00023     url[U("port")] = web::json::value::number(uri.port());
00024
00025     body[U("url")] = url;
00026     body[U("remote_address")] = web::json::value(message.remote_address());
00027
00028     std::vector<web::json::value> matches;
00029     for (size_t i = 0; i < match.size(); i++)
00030     {
00031         std::string m = match.str(i);
00032         matches.push_back(web::json::value::string(utility::conversions::to_string_t(m)));
00033     }
00034     body[U("matches")] = web::json::value::array(matches);
00035
00036     return body;
00037 }
00038
00039 HTTP::Response HTTP::Resource::fallback(Request message, std::smatch match)
00040 {
00041     web::json::value body = request_info(message, match);
00042     Response response;
00043     response.set_body(body);
00044     response.set_status_code(Status::NotImplemented);
00045     this->apply_headers(response);
00046     return response;
00047 }
00048
00049 void HTTP::Resource::apply_headers(Response & response)
00050 {
00051     utility::string_t allowed_methods;
00052     for (auto i = this->allowed_methods.begin(); i != this->allowed_methods.end(); i++)
00053     {
00054         if (i != this->allowed_methods.begin())
00055         {
00056             allowed_methods.append(U(", "));
00057         }
00058         allowed_methods.append(*i);
00059     }
00060     response.headers().add(U("Access-Control-Allow-Origin"),
00061         utility::conversions::to_string_t(allowed_origins));
00062     response.headers().add(U("Access-Control-Allow-Headers"), U("content-type"));
00063     response.headers().add(U("Allow"), allowed_methods);
00064     response.headers().add(U("Content-Type"), utility::conversions::to_string_t(content_type));
00065 }
00066
00067 HTTP::Resource::Resource()
00068 {
00069     allowed_origins = "*";
00070     allowed_methods = { web::http::methods::GET, web::http::methods::DEL, web::http::methods::PATCH,
00071         web::http::methods::POST, web::http::methods::OPTIONS, web::http::methods::HEAD,
00072         web::http::methods::PUT };
00073     content_type = "application/json";
00074 }
00075
00076 HTTP::Resource::~Resource()
00077 {
00078 }
00079
00080 HTTP::Response HTTP::Resource::handle(Request message, std::smatch match)

```



```

00079 {
00080     Response response;
00081     try
00082     {
00083         if (message.method() == Methods::GET)
00084         {
00085             response = this->handle_get(message, match);
00086         }
00087         else if (message.method() == Methods::POST)
00088         {
00089             response = this->handle_post(message, match);
00090         }
00091         else if (message.method() == Methods::PATCH)
00092         {
00093             response = this->handle_patch(message, match);
00094         }
00095         else if (message.method() == Methods::DEL)
00096         {
00097             response = this->handle_delete(message, match);
00098         }
00099         else if (message.method() == Methods::PUT)
00100         {
00101             response = this->handle_put(message, match);
00102         }
00103         else if (message.method() == Methods::OPTIONS)
00104         {
00105             response = this->handle_options(message, match);
00106         }
00107         else if (message.method() == Methods::HEAD)
00108         {
00109             response = this->handle_head(message, match);
00110         }
00111     }
00112     catch (std::exception& exception)
00113     {
00114         response = handle_exception(message, exception, match);
00115     }
00116     return response;
00117 }
00118 }
00119
00120 HTTP::Response HTTP::Resource::handle_get(Request message, std::smatch match)
00121 {
00122     return this->fallback(message, match);
00123 }
00124
00125 HTTP::Response HTTP::Resource::handle_post(Request message, std::smatch match)
00126 {
00127     return this->fallback(message, match);
00128 }
00129
00130 HTTP::Response HTTP::Resource::handle_patch(Request message, std::smatch match)
00131 {
00132     return this->fallback(message, match);
00133 }
00134
00135
00136 HTTP::Response HTTP::Resource::handle_delete(Request message, std::smatch match)
00137 {
00138     return this->fallback(message, match);
00139 }
00140
00141
00142 HTTP::Response HTTP::Resource::handle_put(Request message, std::smatch match)
00143 {
00144     return this->fallback(message, match);
00145 }
00146
00147
00148 HTTP::Response HTTP::Resource::handle_options(Request message, std::smatch match)
00149 {
00150     return this->fallback(message, match);
00151 }
00152
00153 HTTP::Response HTTP::Resource::handle_head(Request message, std::smatch match)
00154 {
00155     return this->fallback(message, match);
00156 }
00157
00158 HTTP::Response HTTP::Resource::handle_exception(Request message, std::exception & exception,
00159     std::smatch match)
00160 {
00161     web::json::value body = request_info(message, match);
00162     body[U("error")] = web::json::value(utility::conversions::to_string_t(exception.what()));
00163     Response response;
00164     response.set_body(body);
00165     response.set_status_code(Status::InternalServerError);

```

```

00165     this->apply_headers(response);
00166     return response;
00167 }

```

## 9.32 src/REST/Resource.h File Reference

```

#include "constants.h"
#include "Types.h"
#include "cpprest/http_listener.h"
#include <regex>
#include <vector>

```

### Classes

- class [HTTP::Resource](#)  
[HTTP Resource](#) *base class.*

### Namespaces

- namespace [HTTP](#)

## 9.33 Resource.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "Types.h"
00004 #include "cpprest/http_listener.h"
00005 #include <regex>
00006 #include <vector>
00007
00008 namespace HTTP {
00009
00020     class DLL Resource
00021     {
00022     private:
00034         Response fallback(Request message, std::smatch match = std::smatch());
00035     protected:
00042         std::vector<web::http::method> allowed_methods;
00043
00050         std::string content_type;
00051
00060         std::string allowed_origins;
00061
00069         void apply_headers(Response& response);
00070     public:
00076         Resource();
00077
00083         ~Resource();
00084
00096         static web::json::value request_info(Request message, std::smatch match = std::smatch());
00097
00109         virtual Response handle(Request message, std::smatch match = std::smatch());
00110
00122         virtual Response handle_get(Request message, std::smatch match = std::smatch());
00123
00135         virtual Response handle_put(Request message, std::smatch match = std::smatch());
00136
00148         virtual Response handle_post(Request message, std::smatch match = std::smatch());
00149
00161         virtual Response handle_delete(Request message, std::smatch match = std::smatch());
00162
00174         virtual Response handle_patch(Request message, std::smatch match = std::smatch());

```

```

00175
00187     virtual Response handle_options(Request message, std::smatch match = std::smatch());
00188
00200     virtual Response handle_head(Request message, std::smatch match = std::smatch());
00201
00213     virtual Response handle_exception(Request message, std::exception& exception, std::smatch
match = std::smatch());
00214 };
00215 }
00216

```

## 9.34 src/REST/Server.cpp File Reference

```
#include "Server.h"
```

## 9.35 Server.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Server.h"
00002
00003 using namespace std;
00004 using namespace web;
00005 using namespace utility;
00006 using namespace http;
00007 using namespace web::http::experimental::listener;
00008
00009 HTTP::Server::Server(std::string url) : listener/utility::conversions::to_string_t(url)
00010 {
00011     listener.support(std::bind(&Server::handle, this, std::placeholders::_1));
00012 }
00013
00014
00015 HTTP::Server::~Server()
00016 {
00017 }
00018
00019 void HTTP::Server::handle(http_request message)
00020 {
00021     for (auto i = resources.begin(); i != resources.end(); i++)
00022     {
00023         std::smatch match;
00024         std::string line = utility::conversions::to_utf8string(message.relative_uri().to_string());
00025         std::regex regex(i->first);
00026         if (std::regex_search(line, match, regex))
00027         {
00028             Response response = i->second->handle(message, match);
00029             message.reply(response);
00030             return;
00031         }
00032     }
00033     Response response = default_resource.handle(message);
00034     message.reply(response);
00035
00036
00037 }
00038
00039 void HTTP::Server::add(std::string path, std::shared_ptr<Resource> resource)
00040 {
00041     resources.push_back(std::make_pair(path, resource));
00042 }
00043
00044 void HTTP::Server::remove(std::string path)
00045 {
00046     for (auto i = resources.begin(); i != resources.end(); i++)
00047     {
00048         if (i->first == path)
00049         {
00050             resources.erase(i);
00051             break;
00052         }
00053     }
00054 }

```

## 9.36 src/REST/Server.h File Reference

```
#include "constants.h"
#include "cpprest/http_listener.h"
#include "Resource.h"
```

### Classes

- class [HTTP::Server](#)  
*HTTP Server.*

### Namespaces

- namespace [HTTP](#)

## 9.37 Server.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include "cpprest/http_listener.h"
00004 #include "Resource.h"
00005
00006 using namespace web;
00007 using namespace http;
00008 using namespace utility;
00009 using namespace http::experimental::listener;
00010
00011 namespace HTTP
00012 {
00021     class DLL Server
00022     {
00023     private:
00029         Resource default_resource;
00030
00037         std::vector<std::pair<std::string, std::shared_ptr<Resource> > > resources;
00038
00049         void handle(http_request message);
00050
00051
00057         http_listener listener;
00058
00059     public:
00069         Server(std::string url);
00070
00077         ~Server();
00078
00084         inline void open() { listener.open().wait(); }
00085
00091         inline void close() { listener.close().wait(); }
00092
00104         void add(std::string path, std::shared_ptr<Resource> resource);
00105
00114         void remove(std::string path);
00115
00116     };
00117 }
00118
```

## 9.38 src/REST/Types.h File Reference

```
#include <cpprest/http_listener.h>
```

## Namespaces

- namespace [HTTP](#)

## Typedefs

- typedef web::http::http\_request [HTTP::Request](#)  
*[HTTP Request.](#)*
- typedef web::http::http\_response [HTTP::Response](#)  
*[HTTP Response.](#)*
- typedef web::http::status\_codes [HTTP::Status](#)  
*[HTTP Status.](#)*
- typedef web::http::methods [HTTP::Methods](#)  
*[HTTP Methods.](#)*
- typedef web::json::value [HTTP::Json](#)  
*[HTTP JSON.](#)*

## 9.39 Types.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <cpprest/http_listener.h>
00003
00004 namespace HTTP
00005 {
00011     typedef web::http::http_request Request;
00012
00018     typedef web::http::http_response Response;
00019
00025     typedef web::http::status_codes Status;
00026
00032     typedef web::http::methods Methods;
00033
00039     typedef web::json::value Json;
00040
00041 }
```

## 9.40 src/SOIL/Types.h File Reference

```
#include "Time.h"
#include "Enum.h"
```

## Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## Typedefs

- typedef int64\_t [SOIL::INT](#)  
*[SOIL Integer.](#)*
- typedef double [SOIL::DOUBLE](#)  
*[SOIL Double.](#)*
- typedef bool [SOIL::BOOL](#)  
*[SOIL Boolean.](#)*
- typedef std::string [SOIL::STRING](#)  
*[SOIL String.](#)*
- typedef Enum [SOIL::ENUM](#)  
*[SOIL Enum.](#)*
- typedef Time [SOIL::TIME](#)  
*[SOIL Time.](#)*
- typedef std::vector< unsigned int > [SOIL::DIMENSION](#)  
*[SOIL Dimension.](#)*

## 9.41 Types.h

[Go to the documentation of this file.](#)

```

00001
00008 #pragma once
00009 #include "Time.h"
00010 #include "Enum.h"
00011 namespace SOIL
00012 {
00018     typedef int64_t INT;
00019
00026     typedef double DOUBLE;
00027
00033     typedef bool BOOL;
00034
00040     typedef std::string STRING;
00041
00047     typedef Enum ENUM;
00048
00054     typedef Time TIME;
00055
00061     typedef std::vector<unsigned int> DIMENSION;
00062
00063
00064 }
```

## 9.42 src/SIGN/Hasher.cpp File Reference

```

#include "Hasher.h"
#include <openssl/sha.h>
#include <sstream>
#include <iomanip>
```

## 9.43 Hasher.cpp

Go to the documentation of this file.

```

00001 #include "Hasher.h"
00002 #include <openssl/sha.h>
00003 #include <sstream>
00004 #include <iomanip>
00005
00006
00007
00008 std::vector<unsigned char> SIGN::Hasher::sha256(const unsigned char* data, size_t length)
00009 {
00010     SHA256_CTX context;
00011     SHA256_Init(&context);
00012     SHA256_Update(&context, data, length);
00013     std::vector<unsigned char> buffer(SHA256_DIGEST_LENGTH);
00014     SHA256_Final(buffer.data(), &context);
00015     return buffer;
00016 }
00017 }
00018
00019 SIGN::Hasher::Hasher()
00020 {
00021 }
00022
00023
00024 SIGN::Hasher::~Hasher()
00025 {
00026 }
00027
00028 std::vector<unsigned char> SIGN::Hasher::hash()
00029 {
00030     return sha256(data.data(), data.size());
00031 }
00032
00033 void SIGN::Hasher::reset()
00034 {
00035     data.clear();
00036 }
00037
00038 size_t SIGN::Hasher::size(void)
00039 {
00040     return SHA256_DIGEST_LENGTH;
00041 }
00042
00043 std::string SIGN::Hasher::print(std::vector<unsigned char> bytes, bool uppercase)
00044 {
00045     std::ostringstream result;
00046     for (std::string::size_type i = 0; i < bytes.size(); i++)
00047     {
00048         result << std::hex << std::setfill('0') << std::setw(2) << (uppercase ? std::uppercase :
00049         std::nouppercase) << (int)bytes[i];
00050         if (i != bytes.size() - 1)
00051         {
00052             result << std::setw(1) << " ";
00053         }
00054     }
00055     return result.str();
00056 }
00057
00058 }

```

## 9.44 src/SIGN/Hasher.h File Reference

```

#include "constants.h"
#include <vector>
#include <string>

```

### Classes

- class [SIGN::Hasher](#)  
*SHA256 Hasher.*

## Namespaces

- namespace [SIGN](#)

## 9.45 Hasher.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include <vector>
00004 #include <string>
00005
00006 namespace SIGN
00007 {
00014     class DLL Hasher
00015     {
00016     private:
00022         std::vector<unsigned char> data;
00023     public:
00029         Hasher();
00030
00036         ~Hasher();
00037
00047         template <typename T> void push_back(T x);
00048
00057         std::vector<unsigned char> hash();
00058
00064         void reset();
00065
00072         size_t size(void);
00073
00083         static std::string print(std::vector<unsigned char> bytes, bool uppercase = true);
00084
00095         static std::vector<unsigned char> sha256(const unsigned char* data, size_t length);
00096     };
00097
00098     template<typename T>
00099     inline void Hasher::push_back(T x)
00100     {
00101         unsigned char* pointer = reinterpret_cast<unsigned char*>(&x);
00102         for (int i = 0; i < sizeof(T); i++)
00103         {
00104             data.push_back(pointer[i]);
00105         }
00106     }
00107 }

```

## 9.46 src/SIGN/Signer.cpp File Reference

```

#include "Signer.h"
#include <fstream>
#include <sstream>
#include <iostream>
#include <openssl/rsa.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/evp.h>
#include <openssl/opensslv.h>

```

## 9.47 Signer.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Signer.h"

```



```

00002 #include <fstream>
00003 #include <sstream>
00004 #include <iostream>
00005 #include <openssl/rsa.h>
00006 #include <openssl/ssl.h>
00007 #include <openssl/err.h>
00008 #include <openssl/evp.h>
00009 #include <openssl/opensslv.h>
00010
00011 SIGN::Signer::Signer(std::string filename)
00012 {
00013     _name = filename;
00014     std::ifstream keyfile(filename);
00015     std::stringstream buffer;
00016     buffer << keyfile.rdbuf();
00017     std::string key = buffer.str();
00018
00019     BIO* bio = BIO_new(BIO_s_mem());
00020     BIO_write(bio, key.c_str(), static_cast<int>(key.length()));
00021
00022     private_key = PEM_read_bio_PrivateKey(bio, NULL, NULL, NULL);
00023
00024     BIO_free(bio);
00025     context = EVP_MD_CTX_create();
00026     EVP_MD_CTX_set_flags(context, EVP_MD_CTX_FLAG_ONESHOT & EVP_MD_CTX_FLAG_FINALISE &
EVP_MD_CTX_FLAG_REUSE);
00027
00028
00029     // Make sure Signer gets a warm start
00030     std::vector<unsigned char> dummy(256, ' ');
00031     this->sign(dummy);
00032 }
00033
00034
00035
00036 SIGN::Signer::~Signer()
00037 {
00038     EVP_PKEY_free(private_key);
00039 }
00040
00041 std::vector<unsigned char> SIGN::Signer::sign(std::vector<unsigned char> message)
00042 {
00043     EVP_DigestSignInit(context, NULL, EVP_sha256(), NULL, private_key);
00044     EVP_DigestSignUpdate(context, message.data(), message.size());
00045
00046     size_t length = 512;
00047     EVP_DigestSignFinal(context, NULL, &length);
00048     unsigned char* buffer = new unsigned char[length];
00049     EVP_DigestSignFinal(context, buffer, &length);
00050     //EVP_DigestSign(context, buffer, &length, message.data(), message.size());
00051     std::vector<unsigned char> result;
00052     result.reserve(length);
00053     for (unsigned int i = 0; i < length; i++)
00054     {
00055         result.push_back(buffer[i]);
00056     }
00057     delete [] buffer;
00058     return result;
00059 }
00060
00061 std::string SIGN::Signer::openssl_version(void) const
00062 {
00063     return OPENSSL_VERSION_TEXT;
00064 }

```

## 9.48 src/SIGN/Signer.h File Reference

```

#include "constants.h"
#include <vector>
#include <string>

```

### Classes

- class [SIGN::Signer](#)  
*Signer.*

## Namespaces

- namespace [SIGN](#)

## Typedefs

- typedef struct evp\_pkey\_st [EVP\\_PKEY](#)
- typedef struct evp\_md\_ctx\_st [EVP\\_MD\\_CTX](#)

### 9.48.1 Typedef Documentation

#### 9.48.1.1 EVP\_MD\_CTX

typedef struct evp\_md\_ctx\_st [EVP\\_MD\\_CTX](#)

Definition at line 7 of file [Signer.h](#).

#### 9.48.1.2 EVP\_PKEY

typedef struct evp\_pkey\_st [EVP\\_PKEY](#)

Definition at line 6 of file [Signer.h](#).

## 9.49 Signer.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include <vector>
00004 #include <string>
00005
00006 typedef struct evp_pkey_st EVP\_PKEY;
00007 typedef struct evp_md_ctx_st EVP\_MD\_CTX;
00008
00009
00010 namespace SIGN
00011 {
00012     class DLL Signer
00013     {
00014     private:
00026         std::string _name;
00027
00034         EVP\_PKEY* private_key;
00035
00041         EVP\_MD\_CTX* context;
00042     public:
00049         Signer(std::string filename);
00050
00056         ~Signer();
00057
00067         std::vector<unsigned char> sign(std::vector<unsigned char> digest);
00068
00077         std::string openssl_version(void) const;
00078
00086         inline std::string name() const { return _name; }
00087     };
00088 }
00089

```

## 9.50 src/SOIL/Container.cpp File Reference

```
#include "Container.h"
```

## 9.51 Container.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Container.h"
```

## 9.52 src/SOIL/Container.h File Reference

```
#include "constants.h"
#include "REST/Types.h"
#include "json_helpers.h"
#include "Range.h"
```

### Classes

- class [SOIL::Container< T, x, y >](#)  
*Data Container.*
- class [SOIL::Container< T, x, -1 >](#)  
*Template specialization for 1D-Arrays/Vectors.*
- class [SOIL::Container< T, -1, -1 >](#)  
*Template specialization for Scalars.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.53 Container.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include "REST/Types.h"
00004 #include "json_helpers.h"
00005 #include "Range.h"
00006 namespace SOIL
00007 {
00020     template <typename T, int x=-1, int y=-1>
00021     class Container
00022     {
00023     private:
00029         std::vector<std::vector<T>> data;
00030
00037         bool _null;
00038     public:
00046         Container();
00047
```

```

00054         Container(const std::vector<std::vector<T>& value);
00055
00062         Container(HTTP::Json json);
00063
00070         std::vector<std::vector<T> operator*(void) const;
00071
00079         inline bool is_null(void) const {return _null;}
00080
00087         inline void set_null(bool _null = true) { this->_null = _null; }
00088
00097         HTTP::Json wjson(void);
00098
00107         bool check_range(Range<T> range) const;
00108
00117         T& at(int i, int j);
00118
00125         std::vector<unsigned char> serialize_value(void) const;
00126
00133         std::vector<unsigned char> serialize_dimensions(void) const;
00134
00135     };
00136
00145     template <typename T, int x>
00146     class Container<T, x, -1>
00147     {
00148     private:
00149         std::vector<T> data;
00150         bool _null;
00151     public:
00152         Container();
00153         Container(const std::vector<T>& value);
00154         Container(HTTP::Json json);
00155         std::vector<T> operator*(void) const;
00156         inline bool is_null(void) const { return _null; }
00157         inline void set_null(bool _null = true) { this->_null = _null; }
00158         HTTP::Json wjson(void);
00159         bool check_range(Range<T> range) const;
00160         T& at(int i);
00161         std::vector<unsigned char> serialize_value(void) const;
00162         std::vector<unsigned char> serialize_dimensions(void) const;
00163
00164     };
00165
00173     template <typename T>
00174     class Container<T, -1, -1>
00175     {
00176     private:
00177         T data;
00178         bool _null;
00179     public:
00180         Container();
00181         Container(const T& value);
00182         Container(HTTP::Json json);
00183         T operator*(void) const;
00184         inline bool is_null(void) const { return _null; }
00185         inline void set_null(bool _null = true) { this->_null = _null; }
00186         HTTP::Json wjson(void);
00187         bool check_range(Range<T> range) const;
00188         T& at(void);
00189         std::vector<unsigned char> serialize_value(void) const;
00190         std::vector<unsigned char> serialize_dimensions(void) const;
00191
00192     };
00193
00194
00195     template<typename T, int x, int y>
00196     Container<T, x, y>::Container()
00197     {
00198         if ((x < -1) || (y < -1))
00199         {
00200             throw std::runtime_error("Invalid Dimension smaller than -1");
00201         }
00202         if ((x == -1) && (y > -1))
00203         {
00204             throw std::runtime_error("An object can only be one-dimensional in its first dimension!");
00205         }
00206         _null = true;
00207     }
00208
00209     template<typename T, int x, int y>
00210     Container<T, x, y>::Container(const std::vector<std::vector<T>& value)
00211     {
00212         if ((x != 0) && (value.size() != x))
00213         {
00214             throw std::runtime_error("Invalid dimension in assignment!");
00215         }
00216         if (y != 0)

```

```

00217     {
00218         for (auto v : value)
00219         {
00220             if (v.size() != y)
00221             {
00222                 throw std::runtime_error("Invalid dimension in assignment!");
00223             }
00224         }
00225     }
00226     data = value;
00227     _null = false;
00228 }
00229
00230 template<typename T, int x, int y>
00231 Container<T, x, y>::Container(HTTP::Json json)
00232 {
00233     std::vector<std::vector<T>> value;
00234     int i = 0;
00235     for (auto& vx : json.as_array())
00236     {
00237         value.push_back(std::vector<T>());
00238         i++;
00239         for (auto vy : vx.as_array())
00240         {
00241             value.at(i).push_back(to_value<T>(vy));
00242         }
00243     }
00244     if ((x != 0) && (value.size() != x))
00245     {
00246         throw std::runtime_error("Invalid dimension in assignment!");
00247     }
00248     if (y != 0)
00249     {
00250         for (auto v : value)
00251         {
00252             if (v.size() != y)
00253             {
00254                 throw std::runtime_error("Invalid dimension in assignment!");
00255             }
00256         }
00257     }
00258     data = value;
00259     _null = false;
00260 }
00261
00262 template<typename T, int x, int y>
00263 std::vector<std::vector<T>> Container<T, x, y>::operator*(void) const
00264 {
00265     return data;
00266 }
00267
00268 template<typename T, int x, int y>
00269 HTTP::Json Container<T, x, y>::wjson(void)
00270 {
00271     HTTP::Json json_root = HTTP::Json();
00272     json_root[U("dimension")] = HTTP::Json::array({ x, y });
00273     if (is_null())
00274     {
00275         json_root[U("value")] = HTTP::Json::null();
00276     }
00277     else
00278     {
00279         json_root[U("value")] = HTTP::Json::array();
00280         for (int i = 0; i < static_cast<int>(data.size()); i++)
00281         {
00282             json_root[U("value")][i] = HTTP::Json::array();
00283             for (int j = 0; j < static_cast<int>(data.at(i).size()); j++)
00284             {
00285                 json_root[U("value")][i][j] = to_json<T>(data.at(i).at(j));
00286             }
00287         }
00288     }
00289     return json_root;
00290 }
00291
00292 template<typename T, int x, int y>
00293 bool Container<T, x, y>::check_range(Range<T> range) const
00294 {
00295     if (is_null())
00296     {
00297         return true;
00298     }
00299     for (auto vx : data)
00300     {
00301         for (auto vy : vx)
00302         {
00303             if (!range.check(vy))

```

```

00304         {
00305             return false;
00306         }
00307     }
00308 }
00309 return true;
00310 }
00311
00312 template<typename T, int x, int y>
00313 T& Container<T, x, y>::at(int i, int j)
00314 {
00315     if (is_null())
00316     {
00317         throw std::logic_error("Value is NULL");
00318     }
00319     return data.at(i).at(j);
00320 }
00321
00322 template<typename T, int x, int y>
00323 std::vector<unsigned char> Container<T, x, y>::serialize_value(void) const
00324 {
00325     if (is_null())
00326     {
00327         throw std::logic_error("Value is NULL");
00328     }
00329     int current_x = static_cast<int>(data.size());
00330     int current_y = current_x > 0 ? static_cast<int>(data.at(0).size()) : 0;
00331     size_t size = current_x * current_y * sizeof(T);
00332     std::vector<unsigned char> serialization;
00333     serialization.reserve(size);
00334
00335     for (int i = 0; i < current_x ; i++)
00336     {
00337         {
00338             const unsigned char* pointer = reinterpret_cast<const unsigned
00339 char*>(data.at(i).data());
00340             for (int j = 0; j < static_cast<int>(current_y*sizeof(T)); j++)
00341             {
00342                 serialization.push_back(pointer[j]);
00343             }
00344         }
00345     }
00346     return serialization;
00347 }
00348
00349 template<typename T, int x, int y>
00350 std::vector<unsigned char> Container<T, x, y>::serialize_dimensions(void) const
00351 {
00352     size_t size = sizeof(uint8_t) + 2 * sizeof(uint32_t);
00353     std::vector<unsigned char> serialization;
00354     serialization.reserve(size);
00355
00356     uint8_t n_dimensions = 2;
00357     uint32_t current_x = data.size();
00358     uint32_t current_y = current_x > 0 ? data.at(0).size : 0;
00359
00360     serialization.push_back(n_dimensions);
00361
00362     unsigned char* current_x_pointer = reinterpret_cast<unsigned char*>(&current_x);
00363     unsigned char* current_y_pointer = reinterpret_cast<unsigned char*>(&current_y);
00364
00365     for (int i = 0; i < sizeof(uint32_t); i++)
00366     {
00367         serialization.push_back(current_x_pointer[i]);
00368     }
00369     for (int i = 0; i < sizeof(uint32_t); i++)
00370     {
00371         serialization.push_back(current_y_pointer[i]);
00372     }
00373
00374     return serialization;
00375 }
00376
00377 template<typename T, int x>
00378 Container<T, x, -1>::Container()
00379 {
00380     if (x < -1)
00381     {
00382         throw std::runtime_error("Invalid dimension in assignment!");
00383     }
00384     _null = true;
00385 }
00386
00387 template<typename T, int x>
00388 Container<T, x, -1>::Container(const std::vector<T>& value)
00389 {

```

```

00390         if ((x != 0) && (value.size() != x))
00391         {
00392             throw std::runtime_error("Invalid dimension in assignment!");
00393         }
00394         data = value;
00395         _null = false;
00396     }
00397
00398     template<typename T, int x>
00399     Container<T, x, -1>::Container(HTTP::Json json)
00400     {
00401         std::vector<T> value;
00402         for (auto& v : json.as_array())
00403         {
00404             value.push_back(to_value<T>(v));
00405         }
00406         if ((x != 0) && (value.size() != x))
00407         {
00408             throw std::runtime_error("Invalid dimension in assignment!");
00409         }
00410         data = value;
00411         _null = false;
00412     }
00413
00414
00415     template<typename T, int x>
00416     std::vector<T> Container<T, x, -1>::operator*(void) const
00417     {
00418         return data;
00419     }
00420
00421     template<typename T, int x>
00422     HTTP::Json Container<T, x, -1>::wjson(void)
00423     {
00424         HTTP::Json json_root = HTTP::Json();
00425         json_root[U("dimension")] = HTTP::Json::array();
00426         json_root[U("dimension")][0] = HTTP::Json::number(x);
00427         if (is_null())
00428         {
00429             json_root[U("value")] = HTTP::Json::null();
00430         }
00431         else
00432         {
00433             json_root[U("value")] = HTTP::Json::array();
00434             for (int i = 0; i < static_cast<int>(data.size()); i++)
00435             {
00436                 json_root[U("value")][i] = to_json<T>(data.at(i));
00437             }
00438         }
00439         return json_root;
00440     }
00441
00442     template<typename T, int x>
00443     bool Container<T, x, -1>::check_range(Range<T> range) const
00444     {
00445         if (is_null())
00446         {
00447             return true;
00448         }
00449         for (auto vx : data)
00450         {
00451             if (!range.check(vx))
00452             {
00453                 return false;
00454             }
00455         }
00456         return true;
00457     }
00458
00459     template<typename T, int x>
00460     T& Container<T, x, -1>::at(int i)
00461     {
00462         if (is_null())
00463         {
00464             throw std::logic_error("Value is NULL");
00465         }
00466         return data.at(i);
00467     }
00468
00469     template<typename T, int x>
00470     std::vector<unsigned char> Container<T, x, -1>::serialize_value(void) const
00471     {
00472         if (is_null())
00473         {
00474             throw std::logic_error("Value is NULL");
00475         }
00476         int current_x = static_cast<int>(data.size());

```

```

00477         size_t size = current_x * sizeof(T);
00478         std::vector<unsigned char> serialization;
00479         serialization.reserve(size);
00480
00481         const unsigned char* pointer = reinterpret_cast<const unsigned char*>(data.data());
00482         for (int i = 0; i < static_cast<int>(current_x * sizeof(T)); i++)
00483         {
00484             serialization.push_back(pointer[i]);
00485         }
00486         return serialization;
00487     }
00488
00489     template<typename T, int x>
00490     std::vector<unsigned char> Container<T, x, -1>::serialize_dimensions(void) const
00491     {
00492         size_t size = sizeof(uint8_t) + sizeof(uint32_t);
00493         std::vector<unsigned char> serialization;
00494         serialization.reserve(size);
00495
00496         uint8_t n_dimensions = 1;
00497         uint32_t current_x = static_cast<uint32_t>(data.size());
00498
00499         serialization.push_back(n_dimensions);
00500         unsigned char* current_x_pointer = reinterpret_cast<unsigned char*>(&current_x);
00501
00502         for (int i = 0; i < sizeof(uint32_t); i++)
00503         {
00504             serialization.push_back(current_x_pointer[i]);
00505         }
00506         return serialization;
00507     }
00508
00509     template<typename T>
00510     Container<T, -1, -1>::Container()
00511     {
00512         _null = true;
00513     }
00514
00515     template<typename T>
00516     Container<T, -1, -1>::Container(const T& value)
00517     {
00518         data = value;
00519         _null = false;
00520     }
00521
00522     template<typename T>
00523     Container<T, -1, -1>::Container(HTTP::Json json)
00524     {
00525         T value = to_value<T>(json);
00526         data = value;
00527         _null = false;
00528     }
00529
00530     template<typename T>
00531     T Container<T, -1, -1>::operator*(void) const
00532     {
00533         return data;
00534     }
00535
00536     template<typename T>
00537     HTTP::Json Container<T, -1, -1>::wjson(void)
00538     {
00539         HTTP::Json json_root = HTTP::Json();
00540         json_root[U("dimension")] = HTTP::Json::array();
00541         if (is_null())
00542         {
00543             json_root[U("value")] = HTTP::Json::null();
00544         }
00545         else
00546         {
00547             json_root[U("value")] = to_json<T>(data);
00548         }
00549         return json_root;
00550     }
00551
00552     template<typename T>
00553     bool Container<T, -1, -1>::check_range(Range<T> range) const
00554     {
00555         if (is_null())
00556         {
00557             return true;
00558         }
00559         return range.check(data);
00560     }
00561
00562     template<typename T>
00563     T& Container<T, -1, -1>::at(void)

```



```

00564     {
00565         if (is_null())
00566         {
00567             throw std::logic_error("Value is NULL");
00568         }
00569         return data;
00570     }
00571
00572     template<typename T>
00573     std::vector<unsigned char> Container<T, -1, -1>::serialize_value(void) const
00574     {
00575         if (is_null())
00576         {
00577             throw std::logic_error("Value is NULL");
00578         }
00579         size_t size = sizeof(T);
00580         std::vector<unsigned char> serialization;
00581         serialization.reserve(size);
00582
00583         const unsigned char* pointer = reinterpret_cast<const unsigned char*>(&data);
00584         for (int i = 0; i < sizeof(T); i++)
00585         {
00586             serialization.push_back(pointer[i]);
00587         }
00588         return serialization;
00589     }
00590
00591     template<typename T>
00592     std::vector<unsigned char> Container<T, -1, -1>::serialize_dimensions(void) const
00593     {
00594         size_t size = sizeof(uint8_t);
00595         std::vector<unsigned char> serialization;
00596         serialization.reserve(size);
00597
00598         uint8_t n_dimensions = 0;
00599
00600         serialization.push_back(n_dimensions);
00601         return serialization;
00602     }
00603 }

```

## 9.54 src/SOIL/Element.cpp File Reference

```

#include "Element.h"
#include "json_helpers.h"

```

## 9.55 Element.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Element.h"
00002 #include "json_helpers.h"
00003
00004
00005 SOIL::Element::Element(std::shared_ptr<Element> parent, std::string uuid, std::string name,
    std::string description, std::string ontology) : parent(parent), uuid(uuid), name(name),
    description(description), ontology(ontology)
00006 {
00007     if (parent.use_count() > 0)
00008     {
00009         parent->add(uuid, this);
00010         self = (*parent)[uuid];
00011     }
00012     else
00013     {
00014         self.reset(this, &SOIL::null_deleter);
00015     }
00016 }
00017
00018
00019 SOIL::Element::~Element()
00020 {
00021     children.clear();
00022 }

```

```

00023
00024 #pragma warning( push )
00025 #pragma warning( disable : 4717)
00026 std::vector<std::string> SOIL::Element::fqid(void)
00027 {
00028     std::vector<std::string> parent_fqid;
00029     if (parent.use_count() > 0)
00030     {
00031         parent_fqid = parent->fqid();
00032     }
00033     parent_fqid.push_back(uuid);
00034     return parent_fqid;
00035 }
00036 #pragma warning(pop)
00037
00038 std::shared_ptr<SOIL::Element> SOIL::Element::operator[] (std::string fqid)
00039 {
00040     if (fqid == "" || fqid == "/")
00041     {
00042         return self;
00043     }
00044     size_t path_length = fqid.size();
00045     if (fqid[0] == '/')
00046     {
00047         fqid = fqid.substr(1, path_length-1);
00048         path_length--;
00049     }
00050     size_t first_delimiter = fqid.find("/");
00051     if (first_delimiter == std::string::npos)
00052     {
00053         if (children.count(fqid) == 0)
00054         {
00055             if (this->uuid == fqid)
00056             {
00057                 return self;
00058             }
00059             throw std::runtime_error(fqid + " does not match any children of " + uuid);
00060         }
00061         return children[fqid];
00062     }
00063     else
00064     {
00065         std::string dock_off_path = fqid.substr(0, first_delimiter);
00066         std::string continue_path = fqid.substr(first_delimiter + 1, path_length - first_delimiter -
1);
00067         if (children.count(dock_off_path) == 0)
00068         {
00069             if (this->uuid == dock_off_path)
00070             {
00071                 return (*self)[continue_path];
00072             }
00073             throw std::runtime_error(dock_off_path + " does not match any children of " + uuid);
00074         }
00075         return (*children[dock_off_path])[continue_path];
00076     }
00077 }
00078
00079 std::shared_ptr<SOIL::Element> SOIL::Element::add(std::string uuid, std::shared_ptr<Element> child)
00080 {
00081     bool exists = children.count(uuid) > 0;
00082     children[uuid] = child;
00083     return child;
00084 }
00085
00086 std::shared_ptr<SOIL::Element> SOIL::Element::add(std::string uuid, Element * child)
00087 {
00088     return add(uuid, std::shared_ptr<Element>(child));
00089 }
00090
00091 bool SOIL::Element::insert(std::string uuid, std::shared_ptr<Element> child)
00092 {
00093     bool exists = children.count(uuid) > 0;
00094     this->add(uuid, child);
00095     return exists;
00096 }
00097
00098 bool SOIL::Element::insert(std::string uuid, Element* child)
00099 {
00100     return insert(uuid, std::shared_ptr<Element>(child));
00101 }
00102
00103 bool SOIL::Element::remove(std::string uuid)
00104 {
00105     bool exists = children.count(uuid) > 0;
00106     children.erase(uuid);
00107     return exists;
00108 }

```

```

00109
00110 bool SOIL::Element::is_object(void) const
00111 {
00112     return (uuid.substr(0, 3) == "OBJ");
00113 }
00114
00115 bool SOIL::Element::is_variable(void) const
00116 {
00117     return (uuid.substr(0, 3) == "VAR");
00118 }
00119
00120 bool SOIL::Element::is_function(void) const
00121 {
00122     return (uuid.substr(0, 3) == "FUN");
00123 }
00124
00125 bool SOIL::Element::is_parameter(void) const
00126 {
00127     return (uuid.substr(0, 3) == "PAR");
00128 }
00129
00130 HTTP::Json SOIL::Element::wjson(void)
00131 {
00132     HTTP::Json json_root;
00133     json_root[U("uuid")] = SOIL::to_json(uuid);
00134     json_root[U("name")] = SOIL::to_json(name);
00135     json_root[U("description")] = SOIL::to_json(description);
00136     if (ontology == "")
00137     {
00138         json_root[U("ontology")] = HTTP::Json::null();
00139     }
00140     else
00141     {
00142         json_root[U("ontology")] = SOIL::to_json(ontology);
00143     }
00144     return json_root;
00145 }
00146
00147
00148 std::string SOIL::Element::json(void)
00149 {
00150     HTTP::Json content = wjson();
00151     return utility::conversions::to_utf8string(content.serialize());
00152 }
00153
00154
00155 HTTP::Response SOIL::Element::handle(HTTP::Request request, std::smatch match)
00156 {
00157     try
00158     {
00159         std::string path = "";
00160         if (match.size() > 0)
00161         {
00162             path = match.str(0); //utility::conversions::to_utf8string(match.str(0));
00163         }
00164
00165         size_t query_delimiter = path.find("?");
00166         std::shared_ptr<Element> resource = (*self)[path.substr(0, query_delimiter)];
00167
00168         if (resource == self)
00169         {
00170             return Resource::handle(request, match);
00171         }
00172         else
00173         {
00174             return resource->Resource::handle(request, match);
00175         }
00176     }
00177     catch (std::exception& exception)
00178     {
00179         return Resource::handle_exception(request, exception, match);
00180     }
00181 }
00182
00183
00184 }

```

## 9.56 src/SOIL/Element.h File Reference

```

#include "constants.h"
#include "REST/constants.h"

```

```
#include "REST/Resource.h"
#include <memory>
#include <vector>
#include <map>
#include <mutex>
#include <regex>
#include <string>
```

## Classes

- class [SOIL::Element](#)  
*SOIL Base Element.*

## Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## Functions

- void [SOIL::null\\_deleter](#) ([SOIL::Element](#) \*ptr)  
*Null deleter.*

## 9.57 Element.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include "REST/constants.h"
00004 #include "REST/Resource.h"
00005 #include <memory>
00006 #include <vector>
00007 #include <map>
00008 #include <mutex>
00009 #include <regex>
00010 #include <string>
00011
00012 namespace SOIL
00013 {
00023     class DLL Element : public HTTP::Resource
00024     {
00025     private:
00026     public:
00032         std::map<std::string, std::shared_ptr<Element> > children;
00033
00039         std::shared_ptr<Element> parent;
00040
00046         std::shared_ptr<Element> self;
00047
00053         std::string uuid;
00054
00060         std::string name;
00061
00067         std::string description;
00068
00076         std::string ontology;
00077
00083         std::recursive_mutex mutex;
00084
00096         Element(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string ontology = "");
00097
```

```

00105     virtual ~Element();
00106
00113     std::vector<std::string> fqid(void);
00114
00126     std::shared_ptr<Element> operator[] (std::string fqid);
00127
00136     std::shared_ptr<Element> add(std::string uuid, std::shared_ptr<Element> child);
00137
00147     std::shared_ptr<Element> add(std::string uuid, Element* child);
00148
00158     bool insert(std::string uuid, std::shared_ptr<Element> child);
00159
00170     bool insert(std::string uuid, Element* child);
00171
00180     bool remove(std::string uuid);
00181
00191     template<typename T> T* cast(void);
00192
00201     bool is_object(void) const;
00202
00211     bool is_variable(void) const;
00212
00221     bool is_function(void) const;
00222
00231     bool is_parameter(void) const;
00232
00240     virtual HTTP::Json wjson(void);
00241
00249     virtual std::string json(void);
00250
00260     HTTP::Response handle(HTTP::Request request, std::smatch match = std::smatch());
00261
00262 };
00263 template<typename T>
00264 T* Element::cast(void)
00265 {
00266     return dynamic_cast<T*>(self.get());
00267 }
00268
00278 inline void null_deleter(SOIL::Element* ptr) {};
00279
00280 }
00281

```

## 9.58 src/SOIL/Enum.cpp File Reference

```
#include "Enum.h"
```

## 9.59 Enum.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Enum.h"
00002
00003
00004 SOIL::Enum::Enum()
00005 {
00006 }
00007
00008 SOIL::Enum::Enum(std::string value) : _selected(value)
00009 {
00010 }
00011
00012 SOIL::Enum::Enum(std::string value, std::vector<std::string> choices): _selected(value),
    _choices(choices)
00013 {
00014 }
00015
00016
00017 SOIL::Enum::~Enum()
00018 {
00019 }
00020
00021 std::string SOIL::Enum::selected(void) const
00022 {

```

```

00023     return _selected;
00024 }
00025
00026 std::vector<std::string> SOIL::Enum::choices(void)
00027 {
00028     return _choices;
00029 }
00030
00031 int SOIL::Enum::index() const
00032 {
00033     for (int i = 0; i < static_cast<int>(_choices.size()); i++)
00034     {
00035         if (_selected == _choices.at(i))
00036         {
00037             return i;
00038         }
00039     }
00040     throw std::logic_error("Value of ENUM not in choices!");
00041 }
00042
00043 int SOIL::Enum::index(std::string value) const
00044 {
00045     for (int i = 0; i < static_cast<int>(_choices.size()); i++)
00046     {
00047         if (value == _choices.at(i))
00048         {
00049             return i;
00050         }
00051     }
00052     throw std::logic_error("Value of ENUM not in choices!");
00053 }
00054
00055 void SOIL::Enum::set(int value)
00056 {
00057     if (value < static_cast<int>(_choices.size()))
00058     {
00059         _selected = _choices.at(value);
00060     }
00061     else
00062     {
00063         throw std::logic_error("Index not in ENUM choices!");
00064     }
00065 }
00066
00067 void SOIL::Enum::set(std::string value)
00068 {
00069     _selected = value;
00070 }

```

## 9.60 src/SOIL/Enum.h File Reference

```

#include "constants.h"
#include <string>
#include <vector>
#include <stdexcept>

```

### Classes

- class [SOIL::Enum](#)  
*SOIL Enum Datatype.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.61 Enum.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include <string>
00004 #include <vector>
00005 #include <stdexcept>
00006
00007 namespace SOIL
00008 {
00009     class DLL Enum
00010     {
00011     private:
00012         std::string _selected;
00013
00014         std::vector<std::string> _choices;
00015     public:
00016         Enum();
00017
00018         Enum(std::string value);
00019
00020         Enum(std::string value, std::vector<std::string> choices);
00021
00022         ~Enum();
00023
00024         std::string selected(void) const;
00025
00026         std::vector<std::string> choices(void);
00027
00028         int index() const;
00029
00030         int index(std::string value) const;
00031
00032         void set(int value);
00033
00034         void set(std::string value);
00035     };
00036 }

```

## 9.62 src/SOIL/Figure.cpp File Reference

```
#include "Figure.h"
```

### Functions

- [template<> HTTP::Json SOIL::datatype< std::string > \(void\)](#)
- [template<> HTTP::Json SOIL::datatype< SOIL::TIME > \(void\)](#)
- [template<> HTTP::Json SOIL::datatype< SOIL::ENUM > \(void\)](#)

### 9.62.1 Function Documentation

#### 9.62.1.1 SOIL::datatype< SOIL::ENUM >()

```

template<>
HTTP::Json SOIL::datatype< SOIL::ENUM > (
    void )

```

Definition at line 43 of file [Figure.cpp](#).

### 9.62.1.2 SOIL::datatype< SOIL::TIME >()

```
template<>
HTTP::Json SOIL::datatype< SOIL::TIME > (
    void )
```

Definition at line 37 of file [Figure.cpp](#).

### 9.62.1.3 SOIL::datatype< std::string >()

```
template<>
HTTP::Json SOIL::datatype< std::string > (
    void )
```

Definition at line 17 of file [Figure.cpp](#).

## 9.63 Figure.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Figure.h"
00002
00003 template<>
00004 HTTP::Json SOIL::datatype<double>(void)
00005 {
00006     return HTTP::Json::string(U("double"));
00007 }
00008
00009 template<>
00010 HTTP::Json SOIL::datatype<bool>(void)
00011 {
00012     return HTTP::Json::string(U("bool"));
00013 }
00014
00015
00016 template<>
00017 HTTP::Json SOIL::datatype<std::string>(void)
00018 {
00019     return HTTP::Json::string(U("string"));
00020 }
00021
00022
00023 template<>
00024 HTTP::Json SOIL::datatype<int64_t>(void)
00025 {
00026     return HTTP::Json::string(U("int"));
00027 }
00028
00029
00030 template<>
00031 HTTP::Json SOIL::datatype<int>(void)
00032 {
00033     return HTTP::Json::string(U("int"));
00034 }
00035
00036 template<>
00037 HTTP::Json SOIL::datatype<SOIL::TIME>(void)
00038 {
00039     return HTTP::Json::string(U("time"));
00040 }
00041
00042 template<>
00043 HTTP::Json SOIL::datatype<SOIL::ENUM>(void)
00044 {
00045     return HTTP::Json::string(U("enum"));
00046 }
```



## 9.64 src/SOIL/Figure.h File Reference

```
#include "constants.h"
#include "Types.h"
#include "Element.h"
#include "Container.h"
#include "Range.h"
```

### Classes

- class [SOIL::Figure< T, x, y >](#)  
*Intermediate class for [Variable](#) and [Parameter](#) that derives from [Element](#).*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

### Functions

- [template<typename T > HTTP::Json SOIL::datatype \(void\)](#)  
*HTTP JSON datatype.*
- [template<> DLL HTTP::Json SOIL::datatype< double > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< bool > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< std::string > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< int64\\_t > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< int > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< SOIL::TIME > \(void\)](#)
- [template<> DLL HTTP::Json SOIL::datatype< SOIL::ENUM > \(void\)](#)

## 9.65 Figure.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include "Types.h"
00004 #include "Element.h"
00005 #include "Container.h"
00006 #include "Range.h"
00007 namespace SOIL
00008 {
00018     template <typename T>
00019     HTTP::Json datatype(void);
00020     template<>
00021     DLL HTTP::Json datatype<double>(void);
00022     template<>
00023     DLL HTTP::Json datatype<bool>(void);
00024     template<>
00025     DLL HTTP::Json datatype<std::string>(void);
00026     template<>
00027     DLL HTTP::Json datatype<int64_t>(void);
00028     template<>
00029     DLL HTTP::Json datatype<int>(void);
00030     template<>
00031     DLL HTTP::Json datatype<SOIL::TIME>(void);
```

```

00032     template<>
00033     DLL HTTP::Json datatype<SOIL::ENUM>(void);
00034
00035
00047     template <typename T, int x=-1, int y=-1>
00048     class Figure : public Element
00049     {
00050     protected:
00056         TIME time;
00057
00063         std::string unit;
00064
00070         Container<T, x, y> value;
00071
00077         Range<T> range;
00078
00085         virtual void read(void) = 0;
00086
00093         virtual void write(void) = 0;
00094
00095     public:
00109         Figure(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string unit, std::string ontology = "", Range<T> range = Range<T>(), TIME time =
TIME());
00110
00116         ~Figure();
00117
00128         Figure<T, x, y>& operator =(const Container<T, x, y>& value);
00129
00130
00137         Container<T, x, y>& operator* (void);
00138
00147         bool check_range(const Container<T, x, y>& value) const;
00148
00155         void set_range(Range<T> range);
00156
00163         void set_time(TIME time);
00164
00175         void set_value(const Container<T, x, y>& value);
00176
00177
00186         HTTP::Json wjson(void) override;
00187
00198         Container<T, x, y> cast(T value);
00199
00209         virtual void update(const Container<T, x, y>& value, TIME time);
00210     };
00211
00212
00213 }
00214
00215
00216
00217 template<typename T, int x, int y>
00218 SOIL::Figure<T, x, y>::Figure(std::shared_ptr<Element> parent, std::string uuid, std::string name,
std::string description, std::string unit, std::string ontology, Range<T> range, SOIL::TIME time) :
SOIL::Element(parent, uuid, name, description, ontology), unit(unit), range(range), time(time)
00219 {
00220 }
00221
00222 template<typename T, int x, int y>
00223 SOIL::Figure<T,x,y>::~~Figure()
00224 {
00225 }
00226
00227
00228 template<typename T, int x, int y>
00229 HTTP::Json SOIL::Figure<T,x,y>::wjson(void)
00230 {
00231
00232     std::unique_lock<std::recursive_mutex> lock(mutex);
00233     HTTP::Json json_root = SOIL::Element::wjson();
00234     json_root[U("timestamp")] = (time.is_null() || time.rfc3339() == "") ? HTTP::Json::null() :
HTTP::Json::string(utility::conversions::to_string_t(time.rfc3339()));
00235     json_root[U("range")] = range.wjson();
00236     json_root[U("unit")] = (unit.length() == 0) ? HTTP::Json::null() : SOIL::to_json(unit);
00237
00238     HTTP::Json value_json = value.wjson();
00239     json_root[U("value")] = value_json[U("value")];
00240     json_root[U("dimension")] = value_json[U("dimension")];
00241     json_root[U("datatype")] = datatype<T>();
00242
00243
00244     return json_root;
00245 }
00246
00247 template<typename T, int x, int y>

```

```

00248 SOIL::Container<T, x, y> SOIL::Figure<T, x, y>::cast(T value)
00249 {
00250     return Container<T, x, y>(value);
00251 }
00252
00253 template<typename T, int x, int y>
00254 void SOIL::Figure<T,x,y>::read(void)
00255 {
00256 }
00257
00258 template<typename T, int x, int y>
00259 SOIL::Figure<T, x, y>& SOIL::Figure<T,x,y>::operator=(const Container<T, x, y>& value)
00260 {
00261     if (!check_range(value))
00262     {
00263         throw std::range_error("Value is out of range specified for Figure " + uuid);
00264     }
00265     this->value = value;
00266     return *this;
00267 }
00268
00269
00270 template<typename T, int x, int y>
00271 SOIL::Container<T, x, y>& SOIL::Figure<T,x,y>::operator*(void)
00272 {
00273     this->read();
00274     return value;
00275 }
00276
00277 template<typename T, int x, int y>
00278 bool SOIL::Figure<T, x, y>::check_range(const Container<T, x, y>& value) const
00279 {
00280     return value.check_range(range);
00281 }
00282
00283 template<typename T, int x, int y>
00284 void SOIL::Figure<T,x,y>::set_range(Range<T> range)
00285 {
00286     this->range = range;
00287 }
00288
00289
00290 template<typename T, int x, int y>
00291 void SOIL::Figure<T,x,y>::set_time(TIME time)
00292 {
00293     this->time = time;
00294 }
00295
00296
00297 template<typename T, int x, int y>
00298 void SOIL::Figure<T, x, y>::set_value(const Container<T, x, y>& value)
00299 {
00300     if (!check_range(value))
00301     {
00302         throw std::range_error("Value is out of range specified for Figure " + uuid);
00303     }
00304     *this = value;
00305 }
00306
00307 template<typename T, int x, int y>
00308 void SOIL::Figure<T,x,y>::update(const Container<T,x,y>& value, TIME time)
00309 {
00310     std::unique_lock<std::recursive_mutex> lock(mutex);
00311     if (!check_range(value))
00312     {
00313         throw std::range_error("Value is out of range specified for Figure " + uuid);
00314     }
00315     *this = value;
00316     this->time = time;
00317 }
00318
00319
00320
00321
00322

```

## 9.66 src/SOIL/Function.cpp File Reference

```
#include "Function.h"
```

## 9.67 Function.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Function.h"
00002
00003
00004 SOIL::Function::Function(std::shared_ptr<Element> parent, std::string uuid, std::string name,
    std::string description, std::string ontology) : Element(parent, uuid, name, description, ontology)
00005 {
00006     if (uuid.substr(0, 3) != "FUN")
00007     {
00008         throw std::logic_error("UUIDs for functions must start with FUN!");
00009     }
00010
00011     allowed_methods = { HTTP::Methods::GET, HTTP::Methods::POST, HTTP::Methods::OPTIONS,
        HTTP::Methods::HEAD };
00012 }
00013
00014 SOIL::Function::~Function()
00015 {
00016 }
00017
00018 std::shared_ptr<SOIL::Function> SOIL::Function::create(std::shared_ptr<Element> parent, std::string
    uuid, std::string name, std::string description, std::string ontology)
00019 {
00020     Function* function = new Function(parent, uuid, name, description, ontology);
00021     return function->ptr();
00022 }
00023
00024 HTTP::Json SOIL::Function::wjson(void)
00025 {
00026     HTTP::Json json_root = Element::wjson();
00027     json_root[U("arguments")] = HTTP::Json::array();
00028     int i = 0;
00029     for (auto& a : arguments)
00030     {
00031         json_root[U("arguments")][i] = a.second->wjson();
00032         i++;
00033     }
00034
00035     json_root[U("returns")] = HTTP::Json::array();
00036     i = 0;
00037     for (auto& r : returns)
00038     {
00039         json_root[U("returns")][i] = r.second->wjson();
00040         i++;
00041     }
00042     json_root[U("errors")] = HTTP::Json::array();
00043     return json_root;
00044 }
00045
00046 HTTP::Response SOIL::Function::handle_get(HTTP::Request request, std::smatch match)
00047 {
00048     HTTP::Response response;
00049     response.set_body(this->wjson());
00050     response.set_status_code(HTTP::Status::OK);
00051     return response;
00052 }
00053
00054 HTTP::Response SOIL::Function::handle_post(HTTP::Request request, std::smatch match)
00055 {
00056     auto task = request.extract_json();
00057     task.wait();
00058
00059     std::map<std::string, HTTP::Json> arguments;
00060     HTTP::Json body = task.get();
00061
00062     auto json_arguments = body[U("arguments")].as_array();
00063
00064     for (auto i = json_arguments.begin(); i != json_arguments.end(); i++)
00065     {
00066         auto json_argument = i->as_object();
00067         std::string uuid = SOIL::to_value<std::string>(json_argument[U("uuid")]);
00068         HTTP::Json value = json_argument[U("value")];
00069
00070         arguments[uuid] = value;
00071     }
00072
00073     std::map<std::string, HTTP::Json> returns = this->invoke(arguments);
00074
00075     std::vector<web::json::value> json_returns;
00076
00077     for (auto i = returns.begin(); i != returns.end(); i++)
00078     {
00079         web::json::value local;

```

```

00080         local[U("uuid")] = SOIL::to_json<std::string>(i->first);
00081         local[U("value")] = i->second;
00082         json_returns.push_back(local);
00083     }
00084
00085     HTTP::Json json;
00086     json[U("returns")] =HTTP::Json::array(json_returns);
00087
00088     HTTP::Response response;
00089     response.set_body(json);
00090     response.set_status_code(HTTP::Status::OK);
00091
00092     return response;
00093 }
00094
00095 std::map<std::string, HTTP::Json> SOIL::Function::invoke(std::map<std::string, HTTP::Json> arguments)
00096 {
00097     throw std::logic_error("The method has not been implemented!");
00098 }

```

## 9.68 src/SOIL/Function.h File Reference

```

#include "constants.h"
#include "Element.h"
#include "Container.h"
#include "Range.h"
#include "Parameter.h"

```

### Classes

- class [SOIL::Function](#)  
*Function Class.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.69 Function.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "Element.h"
00004 #include "Container.h"
00005 #include "Range.h"
00006 #include "Parameter.h"
00007
00008 namespace SOIL
00009 {
00021     class Function :
00022     public Element
00023     {
00024     private:
00032         std::map<std::string, std::shared_ptr<Element>> arguments;
00033
00041         std::map<std::string, std::shared_ptr<Element>> returns;
00042     public:
00056         DLL Function(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string ontology = "");
00057
00063         DLL ~Function();

```

```

00064
00077     static std::shared_ptr<Function> create(std::shared_ptr<Element> parent, std::string uuid,
std::string name, std::string description, std::string ontology = "");
00078
00094     template<typename T, int x = -1, int y = -1>
00095     void add_argument(std::string uuid, std::string name, std::string description, std::string
unit, SOIL::Range<T> range = SOIL::Range<T>(), std::string ontology = "");
00112     template<typename T, int x = -1, int y = -1>
00113     void add_argument(std::string uuid, std::string name, std::string description, std::string
unit, SOIL::Range<T> range, const Container<T, x, y>& default_value, std::string ontology = "");
00114
00130     template<typename T, int x = -1, int y = -1>
00131     void add_return(std::string uuid, std::string name, std::string description, std::string unit,
SOIL::Range<T> range = SOIL::Range<T>(), std::string ontology = "");
00132
00133
00147     template<typename T, int x = -1, int y = -1>
00148     HTTP::Json make_return(std::string uuid, Container<T,x,y> value);
00149
00163     template<typename T, int x = -1, int y = -1>
00164     Container<T, x, y> make_argument(std::string uuid, HTTP::Json external_json);
00165
00172     DLL HTTP::Json wjson(void);
00173
00183     DLL virtual HTTP::Response handle_get(HTTP::Request request, std::smatch match =
std::smatch());
00184
00196     DLL virtual HTTP::Response handle_post(HTTP::Request request, std::smatch match =
std::smatch());
00197
00215     DLL virtual std::map<std::string, HTTP::Json> invoke(std::map<std::string, HTTP::Json>
arguments);
00216
00223     DLL inline std::shared_ptr<Function> ptr(void)
00224     {
00225         return std::dynamic_pointer_cast<Function>(Element::self);
00226     }
00227 };
00228     template<typename T, int x, int y>
00229     void Function::add_argument(std::string uuid, std::string name, std::string description,
std::string unit, SOIL::Range<T> range, std::string ontology)
00230     {
00231         arguments[uuid].reset(new Parameter<T, x, y>(Element::self, uuid, name, description, unit,
false, ontology, range));
00232     }
00233     template<typename T, int x, int y>
00234     void Function::add_argument(std::string uuid, std::string name, std::string description,
std::string unit, SOIL::Range<T> range, const Container<T,x,y>& default_value, std::string ontology)
00235     {
00236         Parameter<T, x, y>* par = new Parameter<T, x, y>(Element::self, uuid, name, description, unit,
false, ontology, range);
00237         arguments[uuid].reset(par);
00238         *par = default_value;
00239     }
00240
00241
00242     template<typename T, int x,int y>
00243     void Function::add_return(std::string uuid, std::string name, std::string description, std::string
unit, SOIL::Range<T> range, std::string ontology)
00244     {
00245         returns[uuid].reset(new Parameter<T, x, y>(Element::self, uuid, name, description, unit,
false, ontology, range));
00246     }
00247     template<typename T, int x, int y>
00248     HTTP::Json Function::make_return(std::string uuid, Container<T, x, y> value)
00249     {
00250         HTTP::Json json_root = HTTP::Json::object();
00251         json_root[U("uuid")] = to_json<std::string>(uuid);
00252         json_root[U("value")] = value.json()[U("value")];
00253         return json_root;
00254     }
00255     template<typename T, int x, int y>
00256     Container<T, x, y> Function::make_argument(std::string uuid, HTTP::Json external_json)
00257     {
00258         for (auto& a : external_json.as_array())
00259         {
00260             if (to_value<std::string>(a[U("uuid")]) == uuid)
00261             {
00262                 return Container<T, x, y>(a[U("value")]);
00263             }
00264         }
00265         return Container<T, x, y>();
00266     }
00267 }
00268

```

## 9.70 src/SOIL/json\_helpers.cpp File Reference

```
#include "json_helpers.h"
#include "Types.h"
```

### Functions

- `template<> web::json::value SOIL::to_json< std::string > (const std::string &value)`
- `template<> web::json::value SOIL::to_json< SOIL::TIME > (const SOIL::TIME &value)`
- `template<> web::json::value SOIL::to_json< SOIL::ENUM > (const SOIL::ENUM &value)`
- `template<> web::json::value SOIL::to_json< SOIL::BOOL > (const bool &value)`
- `template<> std::string SOIL::to_value< std::string > (web::json::value json)`
- `template<> SOIL::TIME SOIL::to_value< SOIL::TIME > (web::json::value json)`
- `template<> SOIL::ENUM SOIL::to_value< SOIL::ENUM > (web::json::value json)`

### 9.70.1 Function Documentation

#### 9.70.1.1 SOIL::to\_json< SOIL::BOOL >()

```
template<>
web::json::value SOIL::to_json< SOIL::BOOL > (
    const bool & value )
```

Definition at line 47 of file [json\\_helpers.cpp](#).

#### 9.70.1.2 SOIL::to\_json< SOIL::ENUM >()

```
template<>
web::json::value SOIL::to_json< SOIL::ENUM > (
    const SOIL::ENUM & value )
```

Definition at line 41 of file [json\\_helpers.cpp](#).

#### 9.70.1.3 SOIL::to\_json< SOIL::TIME >()

```
template<>
web::json::value SOIL::to_json< SOIL::TIME > (
    const SOIL::TIME & value )
```

Definition at line 35 of file [json\\_helpers.cpp](#).

#### 9.70.1.4 SOIL::to\_json< std::string >()

```
template<>
web::json::value SOIL::to_json< std::string > (
    const std::string & value )
```

Definition at line 29 of file [json\\_helpers.cpp](#).

#### 9.70.1.5 SOIL::to\_value< SOIL::ENUM >()

```
template<>
SOIL::ENUM SOIL::to_value< SOIL::ENUM > (
    web::json::value json )
```

Definition at line 86 of file [json\\_helpers.cpp](#).

#### 9.70.1.6 SOIL::to\_value< SOIL::TIME >()

```
template<>
SOIL::TIME SOIL::to_value< SOIL::TIME > (
    web::json::value json )
```

Definition at line 80 of file [json\\_helpers.cpp](#).

#### 9.70.1.7 SOIL::to\_value< std::string >()

```
template<>
std::string SOIL::to_value< std::string > (
    web::json::value json )
```

Definition at line 74 of file [json\\_helpers.cpp](#).



## 9.71 json\_helpers.cpp

[Go to the documentation of this file.](#)

```

00001 #include "json_helpers.h"
00002 #include "Types.h"
00003
00004 template <typename T>
00005 web::json::value SOIL::to_json(const T& value)
00006 {
00007     return web::json::value(value);
00008 }
00009
00010 template<>
00011 web::json::value SOIL::to_json<double>(const double& value)
00012 {
00013     return web::json::value::number(value);
00014 }
00015
00016 template<>
00017 web::json::value SOIL::to_json<int64_t>(const int64_t& value)
00018 {
00019     return web::json::value::number(value);
00020 }
00021
00022 template<>
00023 web::json::value SOIL::to_json<int>(const int& value)
00024 {
00025     return web::json::value::number(value);
00026 }
00027
00028 template<>
00029 web::json::value SOIL::to_json<std::string>(const std::string& value)
00030 {
00031     return web::json::value::string(utility::conversions::to_string_t(value));
00032 }
00033
00034 template<>
00035 web::json::value SOIL::to_json<SOIL::TIME>(const SOIL::TIME& value)
00036 {
00037     return to_json<std::string>(value.rfc3339());
00038 }
00039
00040 template<>
00041 web::json::value SOIL::to_json<SOIL::ENUM>(const SOIL::ENUM& value)
00042 {
00043     return to_json<std::string>(value.selected());
00044 }
00045
00046 template<>
00047 web::json::value SOIL::to_json<SOIL::BOOL>(const bool& value)
00048 {
00049     return web::json::value::boolean(value);
00050 }
00051
00052
00053
00054 template<>
00055 double SOIL::to_value<double>(web::json::value value)
00056 {
00057     return value.as_double();
00058 }
00059
00060
00061 template<>
00062 int SOIL::to_value<int>(web::json::value value)
00063 {
00064     return value.as_integer();
00065 }
00066
00067 template<>
00068 int64_t SOIL::to_value<int64_t>(web::json::value value)
00069 {
00070     return static_cast<int64_t>(value.as_integer());
00071 }
00072
00073 template<>
00074 std::string SOIL::to_value<std::string>(web::json::value json)
00075 {
00076     return utility::conversions::to_utf8string(json.as_string());
00077 }
00078
00079 template<>
00080 SOIL::TIME SOIL::to_value<SOIL::TIME>(web::json::value json)
00081 {
00082     return SOIL::TIME(to_value<std::string>(json));

```

```

00083 }
00084
00085 template<>
00086 SOIL::ENUM SOIL::to_value<SOIL::ENUM>(web::json::value json)
00087 {
00088     return SOIL::ENUM(to_value<std::string>(json));
00089 }
00090
00091 template<>
00092 bool SOIL::to_value<bool>(web::json::value json)
00093 {
00094     return json.as_bool();
00095 }

```

## 9.72 src/SOIL/json\_helpers.h File Reference

```

#include "constants.h"
#include "cpprest/http_listener.h"
#include "Types.h"

```

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

### Functions

- template<typename T >  
DLL web::json::value [SOIL::to\\_json](#) (const T &value)  
*Value to JSON.*
- template<typename T >  
DLL T [SOIL::to\\_value](#) (web::json::value json)  
*JSON to Value.*
- template<> DLL web::json::value [SOIL::to\\_json< double >](#) (const double &value)
- template<> DLL web::json::value [SOIL::to\\_json< int64\\_t >](#) (const int64\_t &value)
- template<> DLL web::json::value [SOIL::to\\_json< int >](#) (const int &value)
- template<> DLL web::json::value [SOIL::to\\_json< std::string >](#) (const std::string &value)
- template<> DLL web::json::value [SOIL::to\\_json< SOIL::TIME >](#) (const [SOIL::TIME](#) &value)
- template<> DLL web::json::value [SOIL::to\\_json< SOIL::ENUM >](#) (const [SOIL::ENUM](#) &value)
- template<> DLL web::json::value [SOIL::to\\_json< SOIL::BOOL >](#) (const bool &value)
- template<> DLL double [SOIL::to\\_value< double >](#) (web::json::value value)
- template<> DLL int [SOIL::to\\_value< int >](#) (web::json::value value)
- template<> DLL int64\_t [SOIL::to\\_value< int64\\_t >](#) (web::json::value value)
- template<> DLL std::string [SOIL::to\\_value< std::string >](#) (web::json::value json)
- template<> DLL [SOIL::TIME](#) [SOIL::to\\_value< SOIL::TIME >](#) (web::json::value json)
- template<> DLL [SOIL::ENUM](#) [SOIL::to\\_value< SOIL::ENUM >](#) (web::json::value json)
- template<> DLL bool [SOIL::to\\_value< bool >](#) (web::json::value json)

## 9.73 json\_helpers.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "cpprest/http_listener.h"
00004 #include "Types.h"
00005
00006 namespace SOIL
00007 {
00018     template <typename T>
00019     DLL web::json::value to_json(const T& value);
00020
00031     template <typename T>
00032     DLL T to_value(web::json::value json);
00033
00034
00035     template<>
00036     DLL web::json::value to_json<double>(const double& value);
00037     template<>
00038     DLL web::json::value to_json<int64_t>(const int64_t& value);
00039     template<>
00040     DLL web::json::value to_json<int>(const int& value);
00041     template<>
00042     DLL web::json::value to_json<std::string>(const std::string& value);
00043     template<>
00044     DLL web::json::value to_json<SOIL::TIME>(const SOIL::TIME& value);
00045     template<>
00046     DLL web::json::value to_json<SOIL::ENUM>(const SOIL::ENUM& value);
00047     template<>
00048     DLL web::json::value to_json<SOIL::BOOL>(const bool& value);
00049     template<>
00050     DLL double to_value<double>(web::json::value value);
00051     template<>
00052     DLL int to_value<int>(web::json::value value);
00053     template<>
00054     DLL int64_t to_value<int64_t>(web::json::value value);
00055     template<>
00056     DLL std::string to_value<std::string>(web::json::value json);
00057     template<>
00058     DLL SOIL::TIME to_value<SOIL::TIME>(web::json::value json);
00059     template<>
00060     DLL SOIL::ENUM to_value<SOIL::ENUM>(web::json::value json);
00061     template<>
00062     DLL bool to_value<bool>(web::json::value json);
00063 }

```

## 9.74 src/SOIL/Object.cpp File Reference

```
#include "Object.h"
```

## 9.75 Object.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Object.h"
00002
00003
00004 SOIL::Object::Object(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string ontology) : Element(parent, uuid, name, description, ontology)
00005 {
00006     if (uuid.substr(0, 3) != "OBJ")
00007     {
00008         throw std::logic_error("UUIDs for objects must start with OBJ!");
00009     }
00010     allowed_methods = { HTTP::Methods::GET, HTTP::Methods::OPTIONS, HTTP::Methods::HEAD,
HTTP::Methods::PUT };
00011 }
00012
00013
00014 SOIL::Object::~Object()
00015 {

```

```

00016 }
00017
00018 std::shared_ptr<SOIL::Object> SOIL::Object::create(std::shared_ptr<Element> parent, std::string uuid,
std::string name, std::string description, std::string ontology)
00019 {
00020     Object* object = new Object(parent, uuid, name, description, ontology);
00021     return object->ptr();
00022 }
00023
00024 HTTP::Json SOIL::Object::wjson(void)
00025 {
00026
00027     HTTP::Json json_root = Element::wjson();
00028     json_root[U("children")] = HTTP::Json::array();
00029     int i = 0;
00030     for (auto& child : children)
00031     {
00032         json_root[U("children")][i] = HTTP::Json();
00033         json_root[U("children")][i][U("uuid")] =
HTTP::Json::string(utility::conversions::to_string_t(child.first));
00034         json_root[U("children")][i][U("name")] =
HTTP::Json::string(utility::conversions::to_string_t(child.second->name));
00035         i++;
00036     }
00037     return json_root;
00038 }
00039
00040 void SOIL::Object::read(void)
00041 {
00042 }
00043
00044 HTTP::Json SOIL::Object::insert(HTTP::Json body)
00045 {
00046     throw std::logic_error("This function has not been implemented!");
00047 }
00048
00049 void SOIL::Object::remove(void)
00050 {
00051 }
00052 }
00053
00054 HTTP::Response SOIL::Object::handle_get(HTTP::Request message, std::smatch match)
00055 {
00056     this->read();
00057     HTTP::Response response;
00058     response.set_body(this->wjson());
00059     response.set_status_code(HTTP::Status::OK);
00060     return response;
00061 }
00062
00063 HTTP::Response SOIL::Object::handle_put(HTTP::Request message, std::smatch match)
00064 {
00065     auto task = message.extract_json();
00066     task.wait();
00067     HTTP::Json body = task.get();
00068     HTTP::Json json = this->insert(body);
00069     HTTP::Response response;
00070     response.set_body(json);
00071     response.set_status_code(HTTP::Status::Created);
00072     return response;
00073 }
00074
00075 HTTP::Response SOIL::Object::handle_delete(HTTP::Request message, std::smatch match)
00076 {
00077     this->remove();
00078     parent->remove(this->uuid);
00079     HTTP::Response response;
00080     response.set_status_code(HTTP::Status::OK);
00081     return response;
00082 }

```

## 9.76 src/SOIL/Object.h File Reference

```

#include "constants.h"
#include "Element.h"

```

## Classes

- class [SOIL::Object](#)  
*Object Class.*

## Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.77 Object.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "Element.h"
00004
00005 namespace SOIL
00006 {
00018     class DLL Object :
00019         public Element
00020     {
00021     public:
00034         Object(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string ontology = "");
00035
00042         ~Object();
00043
00057         static std::shared_ptr<Object> create(std::shared_ptr<Element> parent, std::string uuid,
std::string name, std::string description, std::string ontology = "");
00058
00065         web::json::value wjson(void);
00066
00073         virtual void read(void);
00074
00087         virtual HTTP::Json insert(HTTP::Json body);
00088
00095         virtual void remove(void);
00096
00108         HTTP::Response handle_get(HTTP::Request message, std::smatch match = std::smatch()) override;
00109
00124         HTTP::Response handle_put(HTTP::Request message, std::smatch match = std::smatch()) override;
00125
00141         HTTP::Response handle_delete(HTTP::Request message, std::smatch match = std::smatch())
override;
00142
00149         inline std::shared_ptr<Object> ptr(void)
00150         {
00151             return std::dynamic_pointer_cast<Object>(Element::self);
00152         }
00153     };
00154 }
00155

```

## 9.78 src/SOIL/Parameter.cpp File Reference

```
#include "Parameter.h"
```

## 9.79 Parameter.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Parameter.h"
```

## 9.80 src/SOIL/Parameter.h File Reference

```
#include "constants.h"
#include "Types.h"
#include "Element.h"
#include "Container.h"
#include "Range.h"
#include "Figure.h"
#include "MQTT/Publisher.h"
#include <boost/algorithm/string/join.hpp>
```

### Classes

- class [SOIL::Parameter< T, x, y >](#)  
*Parameter Class.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.81 Parameter.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include "Types.h"
00004 #include "Element.h"
00005 #include "Container.h"
00006 #include "Range.h"
00007 #include "Figure.h"
00008 #include "MQTT/Publisher.h"
00009 #include <boost/algorithm/string/join.hpp>
00010
00011 namespace SOIL
00012 {
00033     template <typename T, int x=-1, int y=-1>
00034     class Parameter : public Figure<T, x, y>
00035     {
00036     protected:
00037
00038         //Looks stale as Figure as a time member
00039         //TIME time;
00040
00046         bool constant;
00047
00054         virtual void read(void);
00055
00062         virtual void write(void);
00063
00064     public:
00079         Parameter(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string unit, bool constant = false, std::string ontology = "", Range<T> range =
Range<T>(), TIME time = TIME());
00080
00086         ~Parameter();
00087
00103         static std::shared_ptr<Parameter> create(std::shared_ptr<Element> parent, std::string uuid,
std::string name, std::string description, std::string unit, bool constant = false, std::string
ontology = "", Range<T> range = Range<T>(), TIME time = TIME());
00104
00116         Parameter<T, x, y>& operator =(const Container<T, x, y>& value);
00117
00118
```

```

00128         HTTP::Json wjson(void) override;
00129
00141         HTTP::Response handle_get(HTTP::Request message, std::smatch match = std::smatch()) override;
00142
00158         HTTP::Response handle_patch(HTTP::Request message, std::smatch match = std::smatch())
00159         override;
00159
00166         inline std::shared_ptr<Parameter> ptr(void)
00167         {
00168             return std::dynamic_pointer_cast<Parameter>(Element::self);
00169         }
00170
00192         bool mqtt(std::shared_ptr<MQTT::Publisher> publisher, int qos = 0, bool retain = false);
00193     };
00194
00195
00196
00197 }
00198
00199
00200
00201 template<typename T, int x, int y>
00202 SOIL::Parameter<T, x, y>::Parameter(std::shared_ptr<SOIL::Element> parent, std::string uuid,
00203                                     std::string name, std::string description, std::string unit, bool constant, std::string ontology,
00204                                     Range<T> range, SOIL::TIME time) : SOIL::Figure<T, x, y>(parent, uuid, name, description, unit,
00205                                     ontology, range, time), constant(constant)
00206 {
00207     if (uuid.substr(0, 3) != "PAR")
00208     {
00209         throw std::logic_error("UUIDs for parameters must start with PAR!");
00210     }
00211
00212     HTTP::Resource::allowed_methods = { HTTP::Methods::GET, HTTP::Methods::OPTIONS,
00213     HTTP::Methods::HEAD, HTTP::Methods::PATCH };
00214 }
00215
00216 template<typename T, int x, int y>
00217 SOIL::Parameter<T,x,y>::~~Parameter()
00218 {
00219 }
00220
00221 template<typename T, int x, int y>
00222 inline std::shared_ptr<SOIL::Parameter<T, x, y>> >
00223 SOIL::Parameter<T, x, y>::create(std::shared_ptr<Element> parent, std::string uuid, std::string name,
00224                                 std::string description, std::string unit, bool constant, std::string ontology, Range<T> range, TIME
00225                                 time)
00226 {
00227     Parameter<T, x, y>* parameter = new Parameter<T, x, y>(parent, uuid, name, description, unit,
00228     constant, ontology, range, time);
00229     return parameter->ptr();
00230 }
00231
00232 template<typename T, int x, int y>
00233 inline SOIL::Parameter<T, x, y>& SOIL::Parameter<T, x, y>::operator=(const Container<T, x, y>& value)
00234 {
00235     Figure<T,x,y>::operator=(value);
00236     return *this;
00237 }
00238
00239
00240
00241
00242 template<typename T, int x, int y>
00243 HTTP::Json SOIL::Parameter<T,x,y>::wjson(void)
00244 {
00245     std::unique_lock<std::recursive_mutex> lock(Element::mutex);
00246     HTTP::Json json_root = Figure<T,x,y>::wjson();
00247     json_root["constant"] = HTTP::Json::boolean(constant);
00248
00249     return json_root;
00250 }
00251
00252
00253
00254 template<typename T, int x, int y>
00255 void SOIL::Parameter<T,x,y>::read(void)
00256 {
00257 }
00258
00259 template<typename T, int x, int y>
00260 inline void SOIL::Parameter<T, x, y>::write(void)
00261 {
00262 }
00263
00264
00265 template<typename T, int x, int y>
00266 inline HTTP::Response SOIL::Parameter<T, x, y>::handle_get(HTTP::Request message, std::smatch match)
00267 {
00268     this->read();
00269
00270     HTTP::Response response;

```

```

00259     HTTP::Json body = this->wjson();
00260     response.set_body(body);
00261
00262     response.set_status_code(HTTP::Status::OK);
00263
00264     return response;
00265 }
00266
00267 template<typename T, int x, int y>
00268 inline HTTP::Response SOIL::Parameter<T, x, y>::handle_patch(HTTP::Request message, std::smatch match)
00269 {
00270     auto task = message.extract_json();
00271     task.wait();
00272     HTTP::Json external_json = task.get();
00273
00274     SOIL::TIME timestamp;
00275
00276     if (external_json.has_field(U("timestamp")))
00277     {
00278         timestamp = to_value<TIME>(external_json[U("timestamp")]);
00279     }
00280     else
00281     {
00282         timestamp = SOIL::TIME::utc_now();
00283     }
00284
00285     this->update(Container<T, x, y>(external_json[U("value")]), timestamp);
00286
00287     this->write();
00288
00289     HTTP::Response response;
00290     response.set_body(this->wjson());
00291     response.set_status_code(HTTP::Status::Created);
00292
00293     return response;
00294 }
00295
00296 template<typename T, int x, int y>
00297 inline bool SOIL::Parameter<T, x, y>::mqtt(std::shared_ptr<MQTT::Publisher> publisher, int qos, bool
00298 retain)
00299 {
00300     std::string topic = boost::algorithm::join(this->fqid(), "/");
00301     return publisher->publish(topic, this->json(), qos, retain);
00302 }
00303
00304 }
00305
00306
00307

```

## 9.82 src/SOIL/Range.cpp File Reference

```
#include "Range.h"
```

## 9.83 Range.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Range.h"
00002
00003 SOIL::Range<std::string>::Range() : set(false), low(0), high(std::numeric_limits<size_t>::max())
00004 {
00005 }
00006
00007 SOIL::Range<std::string>::Range(size_t low, size_t high) : set(true), low(low), high(high)
00008 {
00009 }
00010
00011 SOIL::Range<std::string>::Range(std::vector<size_t> limits) : set(true)
00012 {
00013     if (limits.size() == 0)
00014     {
00015         set = false;
00016     }

```



```

00017     else if (limits.size() == 2)
00018     {
00019         low = limits.at(0);
00020         high = limits.at(1);
00021         set = true;
00022     }
00023     else
00024     {
00025         throw std::runtime_error("Invalid vector for range initialization!");
00026     }
00027 }
00028
00029 SOIL::Range<std::string>::~Range()
00030 {
00031 }
00032
00033 web::json::value SOIL::Range<std::string>::wjson(void)
00034 {
00035     web::json::value range_array = web::json::value::array();
00036     if (set)
00037     {
00038         range_array[0] = SOIL::to_json<int>(static_cast<int>(low));
00039         range_array[1] = SOIL::to_json<int>(static_cast<int>(high));
00040     }
00041     return range_array;
00042 }
00043
00044 bool SOIL::Range<std::string>::check(const std::string & value)
00045 {
00046     if (set)
00047     {
00048         return ((value.size() >= low) && (value.size() <= high));
00049     }
00050     else
00051     {
00052         return true;
00053     }
00054 }
00055
00056
00057
00058 SOIL::Range<SOIL::ENUM>::Range() : set(false)
00059 {
00060 }
00061
00062
00063 SOIL::Range<SOIL::ENUM>::Range(std::vector<std::string> choices) : set(true), choices(choices)
00064 {
00065 }
00066 }
00067
00068 SOIL::Range<SOIL::ENUM>::~Range()
00069 {
00070 }
00071
00072 web::json::value SOIL::Range<SOIL::ENUM>::wjson(void)
00073 {
00074     web::json::value range_array = web::json::value::array();
00075     if (set)
00076     {
00077         for (int i = 0; i < static_cast<int>(choices.size()); i++)
00078         {
00079             range_array[i] = SOIL::to_json<std::string>(choices.at(i));
00080         }
00081     }
00082     return range_array;
00083 }
00084
00085 bool SOIL::Range<SOIL::ENUM>::check(const std::string& value)
00086 {
00087     if (set)
00088     {
00089         for (int i = 0; i < static_cast<int>(choices.size()); i++)
00090         {
00091             if (value == choices.at(i))
00092             {
00093                 return true;
00094             }
00095         }
00096         return false;
00097     }
00098     else
00099     {
00100         return true;
00101     }
00102 }
00103

```

```

00104 bool SOIL::Range<SOIL::ENUM>::check(const SOIL::ENUM& value)
00105 {
00106     return this->check(value.selected());
00107 }
00108

```

## 9.84 src/SOIL/Range.h File Reference

```

#include "constants.h"
#include "json_helpers.h"
#include "Types.h"
#include "REST/Types.h"

```

### Classes

- class [SOIL::Range< T >](#)  
*Range Helper Class.*
- class [SOIL::Range< std::string >](#)  
*String Range.*
- class [SOIL::Range< ENUM >](#)  
*Enum Range.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.85 Range.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "json_helpers.h"
00004 #include "Types.h"
00005 #include "REST/Types.h"
00006
00007 namespace SOIL
00008 {
00023     template <typename T>
00024     class Range
00025     {
00026     private:
00032         T low;
00033
00039         T high;
00040
00047         bool set;
00048     public:
00054         Range();
00055
00064         Range(T low, T high);
00065
00074         Range(std::vector<T> limits);
00075
00081         ~Range();
00082
00089         HTTP::Json wjson(void);
00090
00101         bool check(const T & value);

```

```

00102     };
00103
00109     template<>
00110     class DLL Range<std::string>
00111     {
00112     private:
00113         size_t low;
00114         size_t high;
00115         bool set;
00116     public:
00117         Range();
00118         Range(size_t low, size_t high);
00119         Range(std::vector<size_t> limits);
00120         ~Range();
00121         HTTP::Json wjson(void);
00122         bool check(const std::string& value);
00123     };
00124
00130     template<>
00131     class DLL Range<ENUM>
00132     {
00133     private:
00134         std::vector<std::string> choices;
00135         bool set;
00136     public:
00137         Range();
00138         Range(std::vector<std::string> choices);
00139         ~Range();
00140         HTTP::Json wjson(void);
00141         bool check(const std::string& value);
00142         bool check(const SOIL::ENUM& value);
00143     };
00144
00145     template<typename T>
00146     Range<T>::Range() : set(false)
00147     {
00148     }
00149
00150     template<typename T>
00151     Range<T>::Range(T low, T high): set(true), low(low), high(high)
00152     {
00153     }
00154
00155     template<typename T>
00156     Range<T>::Range(std::vector<T> limits)
00157     {
00158         if (limits.size() == 0)
00159         {
00160             set = false;
00161         }
00162         else if (limits.size() == 2)
00163         {
00164             low = limits.at(0);
00165             high = limits.at(1);
00166             set = true;
00167         }
00168         else
00169         {
00170             throw std::runtime_error("Invalid vector for range initialization!");
00171         }
00172     }
00173
00174     template<typename T>
00175     Range<T>::~~Range()
00176     {
00177     }
00178
00179     template<typename T>
00180     HTTP::Json Range<T>::wjson(void)
00181     {
00182         HTTP::Json range_array = HTTP::Json::array();
00183         if (set)
00184         {
00185             range_array[0] = to_json<T>(low);
00186             range_array[1] = to_json<T>(high);
00187         }
00188         return range_array;
00189     }
00190
00191     template<typename T>
00192     bool Range<T>::check(const T & value)
00193     {
00194         if (set)
00195         {
00196             return ((value >= low) && (value <= high));
00197         }
00198     }

```

```

00199         else
00200         {
00201             return true;
00202         }
00203     }
00204
00205
00206
00207
00208 }
00209
00210

```

## 9.86 src/SOIL/Time.cpp File Reference

```

#include "Time.h"
#include <boost/algorithm/string/replace.hpp>

```

## 9.87 Time.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Time.h"
00002 #include <boost/algorithm/string/replace.hpp>
00003
00004
00005
00006 std::string SOIL::Time::rfc3339(void) const
00007 {
00008     if (is_null())
00009     {
00010         return "";
00011     }
00012     else
00013     {
00014         std::string timestring = boost::posix_time::to_iso_extended_string(timestamp);
00015         boost::replace_all(timestring, " ", ".");
00016         if (timestring == "not-a-date-time")
00017         {
00018             return "";
00019         }
00020         else
00021         {
00022             return timestring + "Z";
00023         }
00024     }
00025 }
00026
00027 SOIL::Time::Time()
00028 {
00029 }
00030
00031 SOIL::Time::Time(std::string value)
00032 {
00033     if (value == "")
00034     {
00035         _null = true;
00036     }
00037     else
00038     {
00039         boost::replace_all(value, "T", " ");
00040         boost::replace_all(value, "Z", "");
00041         timestamp = boost::posix_time::time_from_string(value);
00042         _null = false;
00043     }
00044 }
00045
00046 SOIL::Time::Time(boost::posix_time::ptime value)
00047 {
00048     _null = false;
00049     timestamp = value;
00050 }
00051
00052

```

```

00053 SOIL::Time::~Time()
00054 {
00055 }
00056
00057 boost::posix_time::ptime SOIL::Time::utc_now(void)
00058 {
00059     return boost::posix_time::microsec_clock::universal_time();
00060 }
00061
00062 std::vector<unsigned char> SOIL::Time::serialize(void) const
00063 {
00064     if (is_null())
00065     {
00066         throw std::logic_error("Time is not set");
00067     }
00068     std::vector<unsigned char> serialization;
00069     size_t size = sizeof(uint16_t) + 5 * sizeof(uint8_t) + sizeof(uint32_t);
00070     serialization.reserve(size);
00071
00072     auto date = timestamp.date();
00073     auto time = timestamp.time_of_day();
00074     uint16_t year = static_cast<uint16_t>(date.year());
00075     uint8_t month = static_cast<uint8_t>(date.month().as_number());
00076     uint8_t day = static_cast<uint8_t>(date.day().as_number());
00077     uint8_t hour = static_cast<uint8_t>(time.hours());
00078     uint8_t minute = static_cast<uint8_t>(time.minutes());
00079     uint8_t seconds = static_cast<uint8_t>(time.seconds());
00080     uint32_t nanoseconds = static_cast<uint32_t>(time.fractional_seconds());
00081
00082     switch (time.num_fractional_digits())
00083     {
00084     case 3:
00085         nanoseconds *= 1000000;
00086         break;
00087     case 6:
00088         nanoseconds *= 1000;
00089         break;
00090     case 9:
00091         break;
00092     default:
00093         throw std::logic_error("Unknown fractional seconds!");
00094     }
00095
00096     unsigned char* pointer = reinterpret_cast<unsigned char*>(&year);
00097     for (int i = 0; i < sizeof(uint16_t); i++)
00098     {
00099         serialization.push_back(pointer[i]);
00100     };
00101     serialization.push_back(month);
00102     serialization.push_back(day);
00103     serialization.push_back(hour);
00104     serialization.push_back(minute);
00105     serialization.push_back(seconds);
00106
00107     pointer = reinterpret_cast<unsigned char*>(&nanoseconds);
00108     for (int i = 0; i < sizeof(uint32_t); i++)
00109     {
00110         serialization.push_back(pointer[i]);
00111     };
00112
00113     return serialization;
00114 }
00115
00116 bool SOIL::operator>=(const SOIL::Time & t1, const SOIL::Time & t2)
00117 {
00118     if (t1.is_null() || t2.is_null())
00119     {
00120         return false;
00121     }
00122     else
00123     {
00124         return (t1.timestamp >= t2.timestamp);
00125     }
00126 }
00127
00128 bool SOIL::operator<=(const SOIL::Time & t1, const SOIL::Time & t2)
00129 {
00130     if (t1.is_null() || t2.is_null())
00131     {
00132         return false;
00133     }
00134     else
00135     {
00136         return (t1.timestamp <= t2.timestamp);
00137     }
00138 }

```

## 9.88 src/SOIL/Time.h File Reference

```
#include "constants.h"
#include <boost/date_time/posix_time/posix_time.hpp>
#include <locale>
```

### Classes

- class [SOIL::Time](#)  
*SOIL Time.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

### Functions

- DLL bool [SOIL::operator>=](#) (const Time &t1, const Time &t2)  
*GEQ Time Operator.*
- DLL bool [SOIL::operator<=](#) (const Time &t1, const Time &t2)  
*LEQ Time Operator.*

## 9.89 Time.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include <boost/date_time/posix_time/posix_time.hpp>
00004 #include <locale>
00005 namespace SOIL
00006 {
00012     class Time
00013     {
00014     private:
00020         bool _null;
00021
00029         boost::posix_time::ptime timestamp;
00030     public:
00039         DLL std::string rfc3339(void) const;
00040
00046         DLL Time();
00047
00058         DLL Time(std::string value);
00059
00068         DLL Time(boost::posix_time::ptime value);
00069
00075         DLL ~Time();
00076
00092         inline bool is_null(void) const { return _null; }
00093
00100         inline void set_null(bool _null = true) { this->_null = _null; }
00101
00111         static DLL boost::posix_time::ptime utc_now(void);
00112
00118         friend DLL bool operator>=(const Time& t1, const Time& t2);
00119
00125         friend DLL bool operator<=(const Time& t1, const Time& t2);
00126
00143         DLL std::vector<unsigned char> serialize(void) const;
00144     };
00145
00158     DLL bool operator>=(const Time& t1, const Time& t2);
00159
00160
00173     DLL bool operator<=(const Time& t1, const Time& t2);
00174 }
```

## 9.90 src/SOIL/Types.cpp File Reference

```
#include "Types.h"
```

### 9.91 Types.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Types.h"
```

## 9.92 src/SOIL/Variable.cpp File Reference

```
#include "Variable.h"
```

### 9.93 Variable.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Variable.h"
```

## 9.94 src/SOIL/Variable.h File Reference

```
#include "constants.h"
#include "Types.h"
#include "Element.h"
#include "Container.h"
#include "Range.h"
#include "Figure.h"
#include "SIGN/Signer.h"
#include "SIGN/Hasher.h"
#include "MQTT/Publisher.h"
#include <boost/algorithm/string/join.hpp>
```

### Classes

- class [SOIL::Variable< T, x, y >](#)  
*Variable Class.*

### Namespaces

- namespace [SOIL](#)  
*Type definitions.*

## 9.95 Variable.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "constants.h"
00003 #include "Types.h"
00004 #include "Element.h"
00005 #include "Container.h"
00006 #include "Range.h"
00007 #include "Figure.h"
00008 #include "SIGN/Signer.h"
00009 #include "SIGN/Hasher.h"
00010 #include "MQTT/Publisher.h"
00011 #include <boost/algorithm/string/join.hpp>
00012
00013
00014 namespace SOIL
00015 {
00016     template <typename T, int x = -1, int y = -1>
00041     class Variable : public Figure<T, x, y>
00042     {
00043     protected:
00044
00051         std::string nonce;
00052
00061         std::vector<unsigned char> hash;
00062
00074         Container<T, x, x> covariance;
00075
00083         virtual void read(void);
00084
00092         virtual void write(void);
00093
00094     public:
00109         Variable(std::shared_ptr<Element> parent, std::string uuid, std::string name, std::string
description, std::string unit, std::string ontology = "", Range<T> range = Range<T>(), TIME time =
TIME(), std::string nonce = "");
00110
00116         ~Variable();
00117
00133         static std::shared_ptr<Variable> create(std::shared_ptr<Element> parent, std::string uuid,
std::string name, std::string description, std::string unit, std::string ontology = "", Range<T>
range = Range<T>(), TIME time = TIME(), std::string nonce = "");
00134
00146         Variable<T, x, y>& operator =(const Container<T, x, y>& value);
00147
00157         HTTP::Json wjson(void) override;
00158
00173         HTTP::Response handle_get(HTTP::Request message, std::smatch match = std::smatch()) override;
00174
00187         HTTP::Response handle_options(HTTP::Request message, std::smatch match = std::smatch())
override;
00188
00199         void update(const Container<T, x, y>& value, TIME time, std::string nonce = "");
00200
00211         void set_covariance(Container<T, x, x> covariance);
00212
00229         std::vector<unsigned char> bytes(void);
00230
00239         std::vector<unsigned char> sha256(void);
00240
00255         std::vector<unsigned char> sign(std::shared_ptr<SIGN::Signer> signer = NULL);
00256
00266         std::vector<unsigned char> fingerprint(std::shared_ptr<SIGN::Signer> signer);
00269
00270
00277         inline std::shared_ptr<Variable> ptr(void)
00278         {
00279             return std::dynamic_pointer_cast<Variable>(Element::self);
00280         }
00281
00303         bool mqtt(std::shared_ptr<MQTT::Publisher> publisher, int qos = 0, bool retain = false);
00304     };
00305
00306
00307 }
00308
00309
00310
00311 template<typename T, int x, int y>
00312 SOIL::Variable<T, x, y>::Variable(std::shared_ptr<Element> parent, std::string uuid, std::string name,
std::string description, std::string unit, std::string ontology, Range<T> range, TIME time,
std::string nonce) : SOIL::Figure<T, x, y>(parent, uuid, name, description, unit, ontology, range,
time)
00313 {

```



```

00314     if (uuid.substr(0, 3) != "VAR")
00315     {
00316         throw std::logic_error("UUIDs for Variables must start with VAR!");
00317     }
00318
00319     HTTP::Resource::allowed_methods = { HTTP::Methods::GET, HTTP::Methods::GET,
    HTTP::Methods::OPTIONS, HTTP::Methods::HEAD};
00320 }
00321
00322 template<typename T, int x, int y>
00323 SOIL::Variable<T, x, y>::~Variable()
00324 {
00325 }
00326
00327 template<typename T, int x, int y>
00328 inline std::shared_ptr<SOIL::Variable<T,x,y> >
    SOIL::Variable<T, x, y>::create(std::shared_ptr<Element> parent, std::string uuid, std::string name,
    std::string description, std::string unit, std::string ontology, Range<T> range, TIME time,
    std::string nonce)
00329 {
00330     Variable<T, x, y>* variable = new Variable<T, x, y>(parent, uuid, name, description, unit,
    ontology, range, time, nonce);
00331     return variable->ptr();
00332 }
00333
00334 template<typename T, int x, int y>
00335 inline SOIL::Variable<T, x, y>& SOIL::Variable<T, x, y>::operator=(const Container<T, x, y>& value)
00336 {
00337     Figure<T, x, y>::operator=(value);
00338     return *this;
00339 }
00340
00341
00342 template<typename T, int x, int y>
00343 HTTP::Json SOIL::Variable<T, x, y>::wjson(void)
00344 {
00345     std::unique_lock<std::recursive_mutex> lock(Element::mutex);
00346     HTTP::Json json_root = SOIL::Figure<T, x, y>::wjson();
00347
00348
00349     std::ostringstream buffer;
00350
00351     for (std::string::size_type i = 0; i < hash.size();i++)
00352     {
00353         buffer << std::hex << std::setfill('0') << std::setw(2) << std::uppercase << (int)hash[i];
00354         if (i != hash.size() - 1)
00355         {
00356             buffer << std::setw(1) << " ";
00357         }
00358     }
00359
00360     std::string readable_hash = buffer.str();
00361
00362     json_root[U("nonce")] = (nonce == "") ? HTTP::Json::null() : SOIL::to_json(nonce);
00363     json_root[U("hash")] = (readable_hash == "") ? HTTP::Json::null() : SOIL::to_json(readable_hash);
00364     json_root[U("covariance")] = covariance.wjson()[U("value")];
00365
00366     return json_root;
00367 }
00368
00369 template<typename T, int x, int y>
00370 void SOIL::Variable<T, x, y>::read(void)
00371 {
00372 }
00373
00374 template<typename T, int x, int y>
00375 void SOIL::Variable<T, x, y>::write(void)
00376 {
00377 }
00378
00379 template<typename T, int x, int y>
00380 inline void SOIL::Variable<T, x, y>::set_covariance(Container<T, x, x> covariance)
00381 {
00382     std::unique_lock<std::recursive_mutex> lock(Element::mutex);
00383     this->covariance = covariance;
00384 }
00385
00386 template<typename T, int x, int y>
00387 inline std::vector<unsigned char> SOIL::Variable<T, x, y>::bytes(void)
00388 {
00389     std::unique_lock<std::recursive_mutex> lock(Element::mutex);
00390     std::vector<unsigned char> result;
00391     std::vector<unsigned char> bytes_dimension = Figure<T, x, y>::value.serialize_dimensions();
00392     std::vector<unsigned char> bytes_value = Figure<T, x, y>::value.serialize_value();
00393     std::vector<unsigned char> bytes_covariance = covariance.serialize_value();
00394     std::vector<unsigned char> bytes_unit(3, ' ');
00395     std::vector<unsigned char> bytes_time = Figure<T,x,y>::time.serialize();

```

```

00396     std::vector<unsigned char> bytes_nonce;
00397     for (int i = 0; i < static_cast<int>(std::min(static_cast<size_t>(3),
Figure<T,x,y>::unit.length())); i++)
00398     {
00399         bytes_unit.at(i) = Figure<T, x, y>::unit.at(i);
00400     }
00401     for (int i = 0; i < static_cast<int>(nonce.length()); i++)
00402     {
00403         bytes_nonce.push_back(nonce.at(i));
00404     }
00405
00406     result.insert(result.end(), bytes_dimension.begin(), bytes_dimension.end());
00407     result.insert(result.end(), bytes_value.begin(), bytes_value.end());
00408     result.insert(result.end(), bytes_covariance.begin(), bytes_covariance.end());
00409     result.insert(result.end(), bytes_unit.begin(), bytes_unit.end());
00410     result.insert(result.end(), bytes_time.begin(), bytes_time.end());
00411     result.insert(result.end(), bytes_nonce.begin(), bytes_nonce.end());
00412
00413     return result;
00414 }
00415
00416 template<typename T, int x, int y>
00417 inline std::vector<unsigned char> SOIL::Variable<T, x, y>::sha256(void)
00418 {
00419     std::vector<unsigned char> data = this->bytes();
00420     return SIGN::Hasher::sha256(data.data(), data.size());
00421 }
00422
00423 template<typename T, int x, int y>
00424 inline std::vector<unsigned char> SOIL::Variable<T, x, y>::fingerprint(std::shared_ptr<SIGN::Signer>
signer)
00425 {
00426     return signer->sign(this->sha256());
00427 }
00428
00429 template<typename T, int x, int y>
00430 inline std::vector<unsigned char> SOIL::Variable<T, x, y>::sign(std::shared_ptr<SIGN::Signer> signer)
00431 {
00432     if (signer != NULL)
00433     {
00434         this->hash = this->fingerprint(signer);
00435     }
00436     else {
00437         this->hash = this->sha256();
00438     }
00439
00440     return this->hash;
00441 }
00442
00443 template<typename T, int x, int y>
00444 inline bool SOIL::Variable<T, x, y>::mqtt(std::shared_ptr<MQTT::Publisher> publisher, int qos, bool
retain)
00445 {
00446     std::string topic = boost::algorithm::join(this->fqid(), "/");
00447     return publisher->publish(topic, this->json(), qos, retain);
00448 }
00449
00450
00451 template<typename T, int x, int y>
00452 inline HTTP::Response SOIL::Variable<T, x, y>::handle_get(HTTP::Request message, std::smatch match)
00453 {
00454     this->read();
00455
00456     HTTP::Response response;
00457     response.set_body(this->wjson());
00458     response.set_status_code(HTTP::Status::OK);
00459
00460     return response;
00461 }
00462
00463 template<typename T, int x, int y>
00464 inline HTTP::Response SOIL::Variable<T, x, y>::handle_options(HTTP::Request message, std::smatch
match)
00465 {
00466     HTTP::Response response;
00467     response.set_body(this->wjson());
00468     response.set_status_code(HTTP::Status::OK);
00469
00470     return response;
00471 }
00472
00473 template<typename T, int x, int y>
00474 inline void SOIL::Variable<T, x, y>::update(const Container<T, x, y>& value, TIME time, std::string
nonce)
00475 {
00476     Figure<T, x, y>::update(value, time);
00477     this->nonce = nonce;

```

```
00478
00479     hash.clear();
00480     covariance.set_null(true);
00481 }
00482
00483
00484
00485
```

## 9.96 src/TEST/main.cpp File Reference

```
#include "MQTT/Publisher.h"
#include "UDP/Broadcast.h"
#include "REST/Server.h"
#include "SOIL/Object.h"
#include "SOIL/Variable.h"
#include "SOIL/Types.h"
#include "SIGN/Hasher.h"
#include "SIGN/Signer.h"
#include <thread>
#include <iostream>
#include <chrono>
```

### Functions

- `int main (int argc, char **argv)`

#### 9.96.1 Function Documentation

##### 9.96.1.1 main()

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

Definition at line 15 of file [main.cpp](#).

## 9.97 main.cpp

[Go to the documentation of this file.](#)

```
00001 #include "MQTT/Publisher.h"
00002 #include "UDP/Broadcast.h"
00003 #include "REST/Server.h"
00004 #include "SOIL/Object.h"
00005 #include "SOIL/Variable.h"
00006 #include "SOIL/Types.h"
00007 #include "SIGN/Hasher.h"
00008 #include "SIGN/Signer.h"
00009 #include <thread>
00010 #include <iostream>
00011 #include <chrono>
```

```

00012
00013
00014
00015 int main(int argc, char** argv)
00016 {
00017     try
00018     {
00019         std::cout << "##### SOIL #####" << std::endl;
00020         std::shared_ptr<SIGN::Signer> signer(new
SIGN::Signer(".././.././assets/private-device-key.pem"));
00021         std::cout << "[INFO] " << signer->openssl_version() << std::endl;
00022         std::shared_ptr<SOIL::Object> root_object = SOIL::Object::create(NULL, "OBJ-ROOT", "Root
Object", "Root Object and entry point for the model.");
00023         std::shared_ptr<SOIL::Object> lsm_object = SOIL::Object::create(root_object, "OBJ-LSM", "LSM
System", "LSM system root object");
00024         std::shared_ptr<SOIL::Object> entities_object = SOIL::Object::create(lsm_object,
"OBJ-Entities", "Mobile Entities", "Mobile entities of the LSM system");
00025         std::shared_ptr<SOIL::Object> target_A123 = SOIL::Object::create(entities_object, "OBJ-A123",
"Target A123", "Target A123 of the LSM system");
00026         std::shared_ptr<SOIL::Variable<double, 3> > variable =
SOIL::Variable<double, 3>::create(target_A123, "VAR-Position", "Position", "A dummy position variable
for testing", "MTR", "");
00027         // NOTE: In above calls, the ptr->() function is used to avoid double allocation of shared
pointer and double life cycle accounting
00028
00029         variable->update(std::vector<double>({ 1.0, 1.5, 2.0 }), SOIL::TIME::utc_now(), "LaVA Test");
00030         variable->set_covariance(std::vector<std::vector<double> >({ { 0.01, 0, 0 }, {0, 0.01, 0}, {0,
0, 0.01 } }));
00031         variable->sign(signer);
00032
00033
00034         std::cout << "[INFO] " << "--- Exemplary JSON Serialization ---" << std::endl;
00035         std::cout << variable->json() << std::endl << std::endl;
00036         std::cout << "[INFO] " << "-----" << std::endl;
00037
00038
00039         std::cout << "##### MQTT #####" << std::endl;
00040         MQTT::Configuration mqtt_config(".././.././assets/mqtt.json");
00041         mqtt_config.certificate_authority = ".././.././assets/MQTT-CA.pem";
00042
00043         std::shared_ptr<MQTT::Publisher> mqtt_publisher =
std::make_shared<MQTT::Publisher>("cpp-test");
00044         mqtt_publisher->configure(mqtt_config);
00045         std::cout << "[INFO] MQTT Connecting to " << mqtt_config.uri() << std::endl;
00046         mqtt_publisher->connect();
00047         std::cout << "[OK] MQTT Connected" << std::endl;
00048
00049         std::cout << "[INFO] Running Loop Test" << std::endl;
00050
00051         auto begin = std::chrono::steady_clock::now();
00052         for (int i = 0; i < 1000; i++)
00053         {
00054             variable->update(std::vector<double>({ 1.0 * i, 2.0 * i, 3.0 * i }),
SOIL::TIME::utc_now(), "Loop Test Update");
00055             variable->set_covariance(std::vector<std::vector<double> >({ { 1.0 * i, 0, 0 }, { 0, 1.0 * i, 0
}, { 0, 0, 1.0 * i } }));
00056             variable->sign(signer);
00057             variable->json();
00058             variable->fqid();
00059             variable->mqtt(mqtt_publisher);
00060         }
00061         auto end = std::chrono::steady_clock::now();
00062         std::cout << "[INFO] " << "Average update, serialization and publish cycle: " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() / 1000.0 << " ms" <<
std::endl << std::endl;
00063
00064         std::this_thread::sleep_for(std::chrono::milliseconds(500));
00065         mqtt_publisher->disconnect();
00066         std::cout << "[OK] MQTT Disconnect" << std::endl << std::endl;
00067         std::this_thread::sleep_for(std::chrono::milliseconds(500));
00068
00069
00070         std::cout << "##### UDP #####" << std::endl;
00071         UDP::Configuration udp_config(".././.././assets/udp.json");
00072         UDP::Broadcast udp_broadcast(1);
00073         udp_broadcast.configure(udp_config);
00074         udp_broadcast.send(variable->json());
00075         std::cout << "[OK] UDP Broadcast sent to localhost on port " << udp_config.clients["127.0.0.1"]
<< std::endl << std::endl;
00076
00077
00078         std::cout << "##### REST #####" << std::endl;
00079         HTTP::Server http_server("http://localhost:8000");
00080         http_server.add("/?(.*)", root_object);
00081         http_server.open();
00082         std::cout << "[OK] HTTP Server listening on http://localhost:8000. Press Enter to quit..." <<
std::endl;

```

```

00083         std::string s;
00084         std::getline(std::cin, s);
00085         http_server.close();
00086         std::cout << "[OK] Closing HTTP Server" << std::endl;
00087
00088         root_object.reset();
00089
00090         return 0;
00091     }
00092     catch (std::exception& e)
00093     {
00094         std::cout << "[ERROR] " << e.what() << std::endl;
00095
00096         return 1;
00097     }
00098 }
00099 }

```

## 9.98 src/UDP/Broadcast.cpp File Reference

```

#include "Broadcast.h"
#include "Exception.h"

```

## 9.99 Broadcast.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Broadcast.h"
00002 #include "Exception.h"
00003
00004
00005 UDP::Broadcast::Broadcast(int n_threads)
00006 {
00007     threads.reset(new boost::thread_group);
00008     work.reset(new boost::asio::io_service::work(io_service));
00009     for (int i = 0; i < n_threads; i++)
00010     {
00011         threads->create_thread(boost::bind(&boost::asio::io_service::run, &io_service));
00012     }
00013     socket.reset(new boost::asio::ip::udp::socket(io_service)); //,
00014     boost::asio::ip::udp::endpoint(boost::asio::ip::udp::v4(), 0));
00015     socket->open(boost::asio::ip::udp::v4());
00016     boost::asio::socket_base::reuse_address reuse_address(true);
00017     socket->set_option(boost::asio::socket_base::reuse_address(true));
00018     //socket->set_option(boost::asio::socket_base::broadcast(true));
00019     endpoints.resize(0);
00020 }
00021
00022
00023 UDP::Broadcast::~Broadcast()
00024 {
00025     if (socket.use_count() > 0)
00026     {
00027         if (socket->is_open())
00028         {
00029             socket->shutdown(boost::asio::ip::udp::socket::shutdown_both);
00030             socket->close();
00031         }
00032         socket.reset();
00033         endpoints.resize(0);
00034     }
00035     work.reset();
00036     threads->join_all();
00037     threads.reset();
00038 }
00039
00040 void UDP::Broadcast::add_client(std::string ip, unsigned int port)
00041 {
00042     std::unique_lock<std::mutex> lock(endpoints_mutex);
00043     try
00044     {
00045         std::vector<boost::asio::ip::udp::endpoint>::iterator it;
00046         boost::asio::ip::address address = boost::asio::ip::address::from_string(ip);

```

```

00047         for (it = endpoints.begin(); it != endpoints.end(); it++)
00048         {
00049             if ((*it).address() == address)
00050             {
00051                 throw UDP::Exception("Client is already among broadcast recipients!");
00052             }
00053         }
00054         endpoints.push_back(boost::asio::ip::udp::endpoint(address, port));
00055     }
00056     catch (std::exception& exception)
00057     {
00058         std::string message = std::string("Error while adding UDP client: ") + exception.what();
00059         throw UDP::Exception(message.c_str());
00060     }
00061 }
00062
00063 void UDP::Broadcast::remove_client(std::string ip)
00064 {
00065     std::unique_lock<std::mutex> lock(endpoints_mutex);
00066     try
00067     {
00068         std::vector<boost::asio::ip::udp::endpoint>::iterator it;
00069         boost::asio::ip::address address = boost::asio::ip::address::from_string(ip);
00070         for (it = endpoints.begin(); it != endpoints.end(); it++)
00071         {
00072             if ((*it).address() == address)
00073             {
00074                 break;
00075             }
00076         }
00077         if (it == endpoints.end())
00078         {
00079             throw UDP::Exception("Client is not in broadcast list!");
00080         }
00081         endpoints.erase(it);
00082     }
00083     catch (std::exception& exception)
00084     {
00085         std::string error_message = std::string("Error while removing UDP client: ") +
exception.what();
00086         throw UDP::Exception(error_message.c_str());
00087     }
00088 }
00089
00090 void UDP::Broadcast::_send(std::string message)
00091 {
00092     if (socket.use_count() == 0)
00093     {
00094         return;
00095     }
00096     if (socket->is_open())
00097     {
00098         std::unique_lock<std::mutex> lock(endpoints_mutex);
00099         std::shared_ptr<std::string> local_message(new std::string(message));
00100         std::vector<boost::asio::ip::udp::endpoint>::iterator it;
00101         for (it = endpoints.begin(); it != endpoints.end(); it++)
00102         {
00103             socket->send_to(boost::asio::buffer(*local_message), *it);
00104             //socket->send_to(boost::asio::buffer(*local_message), *it,
boost::bind(&UDP::Broadcast::handle, this, message, boost::asio::placeholders::error,
boost::asio::placeholders::bytes_transferred));
00105         }
00106     }
00107 }
00108
00109 void UDP::Broadcast::send(std::string message)
00110 {
00111     io_service.post(boost::bind(&UDP::Broadcast::_send, this, message + "\r\n"));
00112 }
00113
00114 void UDP::Broadcast::send(std::vector<std::string> messages)
00115 {
00116     std::string message = "";
00117     for (std::vector<std::string>::iterator it = messages.begin(); it != messages.end(); it++)
00118     {
00119         message += (*it) + "\r\n";
00120     }
00121     io_service.post(boost::bind(&UDP::Broadcast::_send, this, message));
00122 }
00123
00124 /*
00125 void UDP::Broadcast::handle(std::shared_ptr<std::string> message, const boost::system::error_code &
error, std::size_t bytes)
00126 {
00127     std::string error_message = std::string("Error while sending UDP: ") + error.message();
00128     throw UDP::Exception(error_message.c_str());
00129 }

```

```

00130 */
00131
00132 void UDP::Broadcast::configure(UDP::Configuration config)
00133 {
00134     for (UDP::Configuration::iterator it = config.clients.begin(); it != config.clients.end(); it++)
00135     {
00136         this->add_client(it->first, it->second);
00137     }
00138 }

```

## 9.100 src/UDP/Broadcast.h File Reference

```

#include "constants.h"
#include <atomic>
#include <memory>
#include <mutex>
#include <stdexcept>
#include <vector>
#include "Configuration.h"
#include <boost/thread.hpp>
#include <boost/asio.hpp>

```

### Classes

- class [UDP::Broadcast](#)  
*UDP Broadcast.*

### Namespaces

- namespace [UDP](#)

## 9.101 Broadcast.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include "constants.h"
00004 #include <atomic>
00005 #include <memory>
00006 #include <mutex>
00007 #include <stdexcept>
00008 #include <vector>
00009 #include "Configuration.h"
00010 #include <boost/thread.hpp>
00011 #include <boost/asio.hpp>
00012
00013 namespace UDP
00014 {
00022     class DLL Broadcast
00023     {
00024     private:
00031         std::shared_ptr<boost::thread_group> threads;
00032
00038         std::shared_ptr<boost::asio::io_service::work> work;
00039
00045         boost::asio::io_service io_service;
00046
00052         std::shared_ptr<boost::asio::ip::udp::socket> socket;
00053
00059         std::vector<boost::asio::ip::udp::endpoint> endpoints;
00060

```

```

00066         std::mutex endpoints_mutex;
00067
00074         void _send(std::string message);
00075
00076     public:
00086         Broadcast(int n_threads = 1);
00087
00093         ~Broadcast();
00094
00104         void add_client(std::string ip, unsigned int port);
00105
00112         void remove_client(std::string ip);
00113
00120         void send(std::string message);
00121
00128         void send(std::vector<std::string> messages);
00129
00130         //void handle(std::shared_ptr<std::string> message, const boost::system::error_code& error,
std::size_t bytes);
00131
00139         inline int use_count(void) { return static_cast<int>(endpoints.size()); }
00140
00147         void configure(UDP::Configuration config);
00148     };
00149 }
00150

```

## 9.102 src/UDP/Exception.cpp File Reference

```
#include "Exception.h"
```

## 9.103 Exception.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Exception.h"
00002
00003 UDP::Exception::Exception(const char* message) : std::runtime_error(message)
00004 {
00005 }
00006
00007 UDP::Exception::~Exception()
00008 {
00009 }

```

## 9.104 src/UDP/Exception.h File Reference

```
#include "constants.h"
#include <stdexcept>
```

### Classes

- class [UDP::Exception](#)  
*UDP Broadcast Exception.*

### Namespaces

- namespace [UDP](#)



## 9.105 Exception.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "constants.h"
00003 #include <stdexcept>
00004 namespace UDP
00005 {
00011     class DLL Exception :
00012         public std::runtime_error
00013     {
00014     public:
00021         Exception(const char* message = "");
00022
00028         ~Exception(void);
00029
00030     };
00031 }
```



# Index

- \_WIN32\_WINNT
  - constants.h, [149–152](#)
- ~Broadcast
  - UDP::Broadcast, [30](#)
- ~Configuration
  - MQTT::Configuration, [34](#)
  - UDP::Configuration, [39](#)
- ~Element
  - SOIL::Element, [53](#)
- ~Enum
  - SOIL::Enum, [63](#)
- ~Exception
  - MQTT::Exception, [67](#)
  - UDP::Exception, [68](#)
- ~Figure
  - SOIL::Figure< T, x, y >, [71](#)
- ~Function
  - SOIL::Function, [78](#)
- ~Hasher
  - SIGN::Hasher, [85](#)
- ~Object
  - SOIL::Object, [91](#)
- ~Parameter
  - SOIL::Parameter< T, x, y >, [98](#)
- ~Publisher
  - MQTT::Publisher, [104](#)
- ~Range
  - SOIL::Range< ENUM >, [111](#)
  - SOIL::Range< std::string >, [113](#)
  - SOIL::Range< T >, [110](#)
- ~Resource
  - HTTP::Resource, [116](#)
- ~Server
  - HTTP::Server, [123](#)
- ~Signer
  - SIGN::Signer, [125](#)
- ~Time
  - SOIL::Time, [129](#)
- ~Variable
  - SOIL::Variable< T, x, y >, [135](#)
- add
  - HTTP::Server, [123](#)
  - SOIL::Element, [54](#)
- add\_argument
  - SOIL::Function, [78, 79](#)
- add\_client
  - UDP::Broadcast, [30](#)
- add\_return
  - SOIL::Function, [80](#)
- allowed\_methods
  - HTTP::Resource, [121](#)
- allowed\_origins
  - HTTP::Resource, [121](#)
- apply\_headers
  - HTTP::Resource, [116](#)
- at
  - SOIL::Container< T, -1, -1 >, [47](#)
  - SOIL::Container< T, x, -1 >, [50](#)
  - SOIL::Container< T, x, y >, [43](#)
- BOOL
  - SOIL, [19](#)
- Broadcast
  - UDP::Broadcast, [30](#)
- bytes
  - SOIL::Variable< T, x, y >, [135](#)
- cast
  - SOIL::Element, [55](#)
  - SOIL::Figure< T, x, y >, [71](#)
- certificate\_authority
  - MQTT::Configuration, [34](#)
- check
  - SOIL::Range< ENUM >, [112](#)
  - SOIL::Range< std::string >, [113](#)
  - SOIL::Range< T >, [110](#)
- check\_range
  - SOIL::Container< T, -1, -1 >, [47](#)
  - SOIL::Container< T, x, -1 >, [50](#)
  - SOIL::Container< T, x, y >, [43](#)
  - SOIL::Figure< T, x, y >, [72](#)
- children
  - SOIL::Element, [60](#)
- choices
  - SOIL::Enum, [63](#)
- clean\_session
  - MQTT::Configuration, [35](#)
- clients
  - UDP::Configuration, [40](#)
- close
  - HTTP::Server, [124](#)
- code
  - MQTT::Exception, [67](#)
- Configuration
  - MQTT::Configuration, [33](#)
  - UDP::Configuration, [39](#)
- Configuration.cpp
  - json, [145, 147](#)
- configure

- MQTT::Publisher, 104
- UDP::Broadcast, 31
- connect
  - MQTT::Publisher, 105
- connection\_timeout\_s
  - MQTT::Configuration, 35
- constant
  - SOIL::Parameter< T, x, y >, 102
- constants.h
  - \_WIN32\_WINNT, 149–152
- Container
  - SOIL::Container< T, -1, -1 >, 46, 47
  - SOIL::Container< T, x, -1 >, 49
  - SOIL::Container< T, x, y >, 42
- content\_type
  - HTTP::Resource, 121
- covariance
  - SOIL::Variable< T, x, y >, 142
- create
  - SOIL::Function, 80
  - SOIL::Object, 91
  - SOIL::Parameter< T, x, y >, 98
  - SOIL::Variable< T, x, y >, 136
- datatype
  - SOIL, 20
- datatype< bool >
  - SOIL, 21
- datatype< double >
  - SOIL, 21
- datatype< int >
  - SOIL, 21
- datatype< int64\_t >
  - SOIL, 21
- datatype< SOIL::ENUM >
  - SOIL, 21
- datatype< SOIL::TIME >
  - SOIL, 21
- datatype< std::string >
  - SOIL, 22
- description
  - SOIL::Element, 60
- DIMENSION
  - SOIL, 19
- disconnect
  - MQTT::Publisher, 105
- DOUBLE
  - SOIL, 19
- Element
  - SOIL::Element, 53
- ENUM
  - SOIL, 19
- Enum
  - SOIL::Enum, 62, 63
- EVP\_MD\_CTX
  - Signer.h, 170
- EVP\_PKEY
  - Signer.h, 170
- Exception
  - MQTT::Exception, 66
  - UDP::Exception, 68
- Figure
  - SOIL::Figure< T, x, y >, 70
- Figure.cpp
  - SOIL::datatype< SOIL::ENUM >, 183
  - SOIL::datatype< SOIL::TIME >, 183
  - SOIL::datatype< std::string >, 184
- fingerprint
  - SOIL::Variable< T, x, y >, 137
- fqid
  - SOIL::Element, 55
- Function
  - SOIL::Function, 78
- handle
  - HTTP::Resource, 116
  - SOIL::Element, 55
- handle\_delete
  - HTTP::Resource, 117
  - SOIL::Object, 92
- handle\_exception
  - HTTP::Resource, 117
- handle\_get
  - HTTP::Resource, 118
  - SOIL::Function, 81
  - SOIL::Object, 92
  - SOIL::Parameter< T, x, y >, 99
  - SOIL::Variable< T, x, y >, 137
- handle\_head
  - HTTP::Resource, 118
- handle\_options
  - HTTP::Resource, 119
  - SOIL::Variable< T, x, y >, 138
- handle\_patch
  - HTTP::Resource, 119
  - SOIL::Parameter< T, x, y >, 99
- handle\_post
  - HTTP::Resource, 120
  - SOIL::Function, 81
- handle\_put
  - HTTP::Resource, 120
  - SOIL::Object, 93
- hash
  - SIGN::Hasher, 85
  - SOIL::Variable< T, x, y >, 143
- Hasher
  - SIGN::Hasher, 85
- host
  - MQTT::Configuration, 35
- HTTP, 15
  - Json, 15
  - Methods, 15
  - Request, 16
  - Response, 16
  - Status, 16
- HTTP::Resource, 114

- ~Resource, 116
- allowed\_methods, 121
- allowed\_origins, 121
- apply\_headers, 116
- content\_type, 121
- handle, 116
- handle\_delete, 117
- handle\_exception, 117
- handle\_get, 118
- handle\_head, 118
- handle\_options, 119
- handle\_patch, 119
- handle\_post, 120
- handle\_put, 120
- request\_info, 120
- Resource, 115
- HTTP::Server, 122
  - ~Server, 123
  - add, 123
  - close, 124
  - open, 124
  - remove, 124
  - Server, 122
- index
  - SOIL::Enum, 63, 64
- insert
  - SOIL::Element, 56
  - SOIL::Object, 93
- INT
  - SOIL, 19
- invoke
  - SOIL::Function, 82
- is\_connected
  - MQTT::Publisher, 105
- is\_function
  - SOIL::Element, 57
- is\_null
  - SOIL::Container< T, -1, -1 >, 47
  - SOIL::Container< T, x, -1 >, 50
  - SOIL::Container< T, x, y >, 44
  - SOIL::Time, 129
- is\_object
  - SOIL::Element, 57
- is\_parameter
  - SOIL::Element, 57
- is\_variable
  - SOIL::Element, 58
- iterator
  - UDP::Configuration, 39
- Json
  - HTTP, 15
- json
  - Configuration.cpp, 145, 147
  - SOIL::Element, 58
- json\_helpers.cpp
  - SOIL::to\_json< SOIL::BOOL >, 191
  - SOIL::to\_json< SOIL::ENUM >, 191
- SOIL::to\_json< SOIL::TIME >, 191
- SOIL::to\_json< std::string >, 191
- SOIL::to\_value< SOIL::ENUM >, 192
- SOIL::to\_value< SOIL::TIME >, 192
- SOIL::to\_value< std::string >, 192
- keep\_alive
  - MQTT::Configuration, 35
- main
  - main.cpp, 211
- main.cpp
  - main, 211
- make\_argument
  - SOIL::Function, 82
- make\_return
  - SOIL::Function, 83
- message
  - MQTT::MessageContainer, 88
- Methods
  - HTTP, 15
- min\_delay\_ms
  - MQTT::Configuration, 36
  - MQTT::Publisher, 106
- MQTT, 16
- mqtt, 17
  - SOIL::Parameter< T, x, y >, 100
  - SOIL::Variable< T, x, y >, 138
- MQTT::Configuration, 32
  - ~Configuration, 34
  - certificate\_authority, 34
  - clean\_session, 35
  - Configuration, 33
  - connection\_timeout\_s, 35
  - host, 35
  - keep\_alive, 35
  - min\_delay\_ms, 36
  - password, 36
  - path, 36
  - port, 36
  - root, 36
  - ssl, 37
  - uri, 34
  - username, 37
  - verify, 37
  - websocket, 37
- MQTT::Exception, 65
  - ~Exception, 67
  - code, 67
  - Exception, 66
- MQTT::MessageContainer, 88
  - message, 88
  - qos, 88
  - retain, 89
  - topic, 89
- MQTT::Publisher, 103
  - ~Publisher, 104
  - configure, 104
  - connect, 105

- disconnect, 105
- is\_connected, 105
- min\_delay\_ms, 106
- publish, 106
- Publisher, 104
- reconnect, 107
- set\_buffer, 107
- set\_root\_topic, 108
- mutex
  - SOIL::Element, 60
- name
  - SIGN::Signer, 126
  - SOIL::Element, 60
- nonce
  - SOIL::Variable< T, x, y >, 143
- null\_deleter
  - SOIL, 22
- Object
  - SOIL::Object, 90
- ontology
  - SOIL::Element, 60
- open
  - HTTP::Server, 124
- openssl\_version
  - SIGN::Signer, 126
- operator<=
  - SOIL, 22
  - SOIL::Time, 131
- operator>=
  - SOIL, 23
  - SOIL::Time, 131
- operator\*
  - SOIL::Container< T, -1, -1 >, 47
  - SOIL::Container< T, x, -1 >, 50
  - SOIL::Container< T, x, y >, 44
  - SOIL::Figure< T, x, y >, 72
- operator=
  - SOIL::Figure< T, x, y >, 72
  - SOIL::Parameter< T, x, y >, 100
  - SOIL::Variable< T, x, y >, 139
- operator[]
  - SOIL::Element, 58
- PAHO\_MQTT\_IMPORTS
  - Publisher.cpp, 155
- Parameter
  - SOIL::Parameter< T, x, y >, 97
- parent
  - SOIL::Element, 61
- password
  - MQTT::Configuration, 36
- path
  - MQTT::Configuration, 36
- port
  - MQTT::Configuration, 36
- print
  - SIGN::Hasher, 86
- ptr
  - SOIL::Function, 83
  - SOIL::Object, 94
  - SOIL::Parameter< T, x, y >, 101
  - SOIL::Variable< T, x, y >, 139
- publish
  - MQTT::Publisher, 106
- Publisher
  - MQTT::Publisher, 104
- Publisher.cpp
  - PAHO\_MQTT\_IMPORTS, 155
- push\_back
  - SIGN::Hasher, 86
- qos
  - MQTT::MessageContainer, 88
- Range
  - SOIL::Range< ENUM >, 111
  - SOIL::Range< std::string >, 113
  - SOIL::Range< T >, 109
- range
  - SOIL::Figure< T, x, y >, 75
- read
  - SOIL::Figure< T, x, y >, 73
  - SOIL::Object, 94
  - SOIL::Parameter< T, x, y >, 101
  - SOIL::Variable< T, x, y >, 140
- README.md, 145
- reconnect
  - MQTT::Publisher, 107
- remove
  - HTTP::Server, 124
  - SOIL::Element, 59
  - SOIL::Object, 94
- remove\_client
  - UDP::Broadcast, 31
- Request
  - HTTP, 16
- request\_info
  - HTTP::Resource, 120
- reset
  - SIGN::Hasher, 87
- Resource
  - HTTP::Resource, 115
- Response
  - HTTP, 16
- retain
  - MQTT::MessageContainer, 89
- rfc3339
  - SOIL::Time, 129
- root
  - MQTT::Configuration, 36
- selected
  - SOIL::Enum, 64
- self
  - SOIL::Element, 61
- send

- UDP::Broadcast, [31](#)
- serialize
  - SOIL::Time, [130](#)
- serialize\_dimensions
  - SOIL::Container< T, -1, -1 >, [48](#)
  - SOIL::Container< T, x, -1 >, [50](#)
  - SOIL::Container< T, x, y >, [44](#)
- serialize\_value
  - SOIL::Container< T, -1, -1 >, [48](#)
  - SOIL::Container< T, x, -1 >, [51](#)
  - SOIL::Container< T, x, y >, [44](#)
- Server
  - HTTP::Server, [122](#)
- set
  - SOIL::Enum, [64](#), [65](#)
- set\_buffer
  - MQTT::Publisher, [107](#)
- set\_covariance
  - SOIL::Variable< T, x, y >, [140](#)
- set\_null
  - SOIL::Container< T, -1, -1 >, [48](#)
  - SOIL::Container< T, x, -1 >, [51](#)
  - SOIL::Container< T, x, y >, [45](#)
  - SOIL::Time, [130](#)
- set\_range
  - SOIL::Figure< T, x, y >, [73](#)
- set\_root\_topic
  - MQTT::Publisher, [108](#)
- set\_time
  - SOIL::Figure< T, x, y >, [73](#)
- set\_value
  - SOIL::Figure< T, x, y >, [74](#)
- sha256
  - SIGN::Hasher, [87](#)
  - SOIL::Variable< T, x, y >, [140](#)
- SIGN, [17](#)
- sign
  - SIGN::Signer, [126](#)
  - SOIL::Variable< T, x, y >, [141](#)
- SIGN::Hasher, [84](#)
  - ~Hasher, [85](#)
  - hash, [85](#)
  - Hasher, [85](#)
  - print, [86](#)
  - push\_back, [86](#)
  - reset, [87](#)
  - sha256, [87](#)
  - size, [87](#)
- SIGN::Signer, [124](#)
  - ~Signer, [125](#)
  - name, [126](#)
  - openssl\_version, [126](#)
  - sign, [126](#)
  - Signer, [125](#)
- Signer
  - SIGN::Signer, [125](#)
- Signer.h
  - EVP\_MD\_CTX, [170](#)
  - EVP\_PKEY, [170](#)
- size
  - SIGN::Hasher, [87](#)
- SOIL, [17](#)
  - BOOL, [19](#)
  - datatype, [20](#)
  - datatype< bool >, [21](#)
  - datatype< double >, [21](#)
  - datatype< int >, [21](#)
  - datatype< int64\_t >, [21](#)
  - datatype< SOIL::ENUM >, [21](#)
  - datatype< SOIL::TIME >, [21](#)
  - datatype< std::string >, [22](#)
  - DIMENSION, [19](#)
  - DOUBLE, [19](#)
  - ENUM, [19](#)
  - INT, [19](#)
  - null\_deleter, [22](#)
  - operator<=, [22](#)
  - operator>=, [23](#)
  - STRING, [20](#)
  - TIME, [20](#)
  - to\_json, [23](#)
  - to\_json< double >, [24](#)
  - to\_json< int >, [24](#)
  - to\_json< int64\_t >, [24](#)
  - to\_json< SOIL::BOOL >, [24](#)
  - to\_json< SOIL::ENUM >, [24](#)
  - to\_json< SOIL::TIME >, [24](#)
  - to\_json< std::string >, [25](#)
  - to\_value, [25](#)
  - to\_value< bool >, [25](#)
  - to\_value< double >, [25](#)
  - to\_value< int >, [26](#)
  - to\_value< int64\_t >, [26](#)
  - to\_value< SOIL::ENUM >, [26](#)
  - to\_value< SOIL::TIME >, [26](#)
  - to\_value< std::string >, [26](#)
- SOIL::Container< T, -1, -1 >, [46](#)
  - at, [47](#)
  - check\_range, [47](#)
  - Container, [46](#), [47](#)
  - is\_null, [47](#)
  - operator\*, [47](#)
  - serialize\_dimensions, [48](#)
  - serialize\_value, [48](#)
  - set\_null, [48](#)
  - wjson, [48](#)
- SOIL::Container< T, x, -1 >, [48](#)
  - at, [50](#)
  - check\_range, [50](#)
  - Container, [49](#)
  - is\_null, [50](#)
  - operator\*, [50](#)
  - serialize\_dimensions, [50](#)
  - serialize\_value, [51](#)
  - set\_null, [51](#)
  - wjson, [51](#)

- SOIL::Container< T, x, y >, 40
  - at, 43
  - check\_range, 43
  - Container, 42
  - is\_null, 44
  - operator\*, 44
  - serialize\_dimensions, 44
  - serialize\_value, 44
  - set\_null, 45
  - wjson, 45
- SOIL::datatype< SOIL::ENUM >
  - Figure.cpp, 183
- SOIL::datatype< SOIL::TIME >
  - Figure.cpp, 183
- SOIL::datatype< std::string >
  - Figure.cpp, 184
- SOIL::Element, 51
  - ~Element, 53
  - add, 54
  - cast, 55
  - children, 60
  - description, 60
  - Element, 53
  - fqid, 55
  - handle, 55
  - insert, 56
  - is\_function, 57
  - is\_object, 57
  - is\_parameter, 57
  - is\_variable, 58
  - json, 58
  - mutex, 60
  - name, 60
  - ontology, 60
  - operator[], 58
  - parent, 61
  - remove, 59
  - self, 61
  - uuid, 61
  - wjson, 59
- SOIL::Enum, 62
  - ~Enum, 63
  - choices, 63
  - Enum, 62, 63
  - index, 63, 64
  - selected, 64
  - set, 64, 65
- SOIL::Figure< T, x, y >, 69
  - ~Figure, 71
  - cast, 71
  - check\_range, 72
  - Figure, 70
  - operator\*, 72
  - operator=, 72
  - range, 75
  - read, 73
  - set\_range, 73
  - set\_time, 73
  - set\_value, 74
  - time, 75
  - unit, 76
  - update, 74
  - value, 76
  - wjson, 75
  - write, 75
- SOIL::Function, 76
  - ~Function, 78
  - add\_argument, 78, 79
  - add\_return, 80
  - create, 80
  - Function, 78
  - handle\_get, 81
  - handle\_post, 81
  - invoke, 82
  - make\_argument, 82
  - make\_return, 83
  - ptr, 83
  - wjson, 84
- SOIL::Object, 89
  - ~Object, 91
  - create, 91
  - handle\_delete, 92
  - handle\_get, 92
  - handle\_put, 93
  - insert, 93
  - Object, 90
  - ptr, 94
  - read, 94
  - remove, 94
  - wjson, 95
- SOIL::Parameter< T, x, y >, 95
  - ~Parameter, 98
  - constant, 102
  - create, 98
  - handle\_get, 99
  - handle\_patch, 99
  - mqtt, 100
  - operator=, 100
  - Parameter, 97
  - ptr, 101
  - read, 101
  - wjson, 101
  - write, 102
- SOIL::Range< ENUM >, 111
  - ~Range, 111
  - check, 112
  - Range, 111
  - wjson, 112
- SOIL::Range< std::string >, 112
  - ~Range, 113
  - check, 113
  - Range, 113
  - wjson, 114
- SOIL::Range< T >, 108
  - ~Range, 110
  - check, 110



- Range, 109
- wjson, 110
- SOIL::Time, 127
  - ~Time, 129
  - is\_null, 129
  - operator<=, 131
  - operator>=, 131
  - rfc3339, 129
  - serialize, 130
  - set\_null, 130
  - Time, 128
  - utc\_now, 131
- SOIL::to\_json< SOIL::BOOL >
  - json\_helpers.cpp, 191
- SOIL::to\_json< SOIL::ENUM >
  - json\_helpers.cpp, 191
- SOIL::to\_json< SOIL::TIME >
  - json\_helpers.cpp, 191
- SOIL::to\_json< std::string >
  - json\_helpers.cpp, 191
- SOIL::to\_value< SOIL::ENUM >
  - json\_helpers.cpp, 192
- SOIL::to\_value< SOIL::TIME >
  - json\_helpers.cpp, 192
- SOIL::to\_value< std::string >
  - json\_helpers.cpp, 192
- SOIL::Variable< T, x, y >, 132
  - ~Variable, 135
  - bytes, 135
  - covariance, 142
  - create, 136
  - fingerprint, 137
  - handle\_get, 137
  - handle\_options, 138
  - hash, 143
  - mqtt, 138
  - nonce, 143
  - operator=, 139
  - ptr, 139
  - read, 140
  - set\_covariance, 140
  - sha256, 140
  - sign, 141
  - update, 141
  - Variable, 134
  - wjson, 142
  - write, 142
- src/MQTT/Configuration.cpp, 145, 146
- src/MQTT/Configuration.h, 148
- src/MQTT/constants.h, 149, 150
- src/MQTT/LocalException.cpp, 153
- src/MQTT/LocalException.h, 153
- src/MQTT/MessageContainer.h, 154
- src/MQTT/Publisher.cpp, 154, 155
- src/MQTT/Publisher.h, 158
- src/REST/constants.h, 150, 151
- src/REST/Resource.cpp, 159, 160
- src/REST/Resource.h, 162
- src/REST/Server.cpp, 163
- src/REST/Server.h, 164
- src/REST/Types.h, 164, 165
- src/SIGN/constants.h, 151
- src/SIGN/Hasher.cpp, 166, 167
- src/SIGN/Hasher.h, 167, 168
- src/SIGN/Signer.cpp, 168
- src/SIGN/Signer.h, 169, 170
- src/SOIL/constants.h, 152
- src/SOIL/Container.cpp, 171
- src/SOIL/Container.h, 171
- src/SOIL/Element.cpp, 177
- src/SOIL/Element.h, 179, 180
- src/SOIL/Enum.cpp, 181
- src/SOIL/Enum.h, 182, 183
- src/SOIL/Figure.cpp, 183, 184
- src/SOIL/Figure.h, 185
- src/SOIL/Function.cpp, 187, 188
- src/SOIL/Function.h, 189
- src/SOIL/json\_helpers.cpp, 191, 193
- src/SOIL/json\_helpers.h, 194, 195
- src/SOIL/Object.cpp, 195
- src/SOIL/Object.h, 196, 197
- src/SOIL/Parameter.cpp, 197
- src/SOIL/Parameter.h, 198
- src/SOIL/Range.cpp, 200
- src/SOIL/Range.h, 202
- src/SOIL/Time.cpp, 204
- src/SOIL/Time.h, 206
- src/SOIL/Types.cpp, 207
- src/SOIL/Types.h, 165, 166
- src/SOIL/Variable.cpp, 207
- src/SOIL/Variable.h, 207, 208
- src/TEST/main.cpp, 211
- src/UDP/Broadcast.cpp, 213
- src/UDP/Broadcast.h, 215
- src/UDP/Configuration.cpp, 147
- src/UDP/Configuration.h, 149
- src/UDP/constants.h, 152
- src/UDP/Exception.cpp, 216
- src/UDP/Exception.h, 216, 217
- ssl
  - MQTT::Configuration, 37
- Status
  - HTTP, 16
- STRING
  - SOIL, 20
- TIME
  - SOIL, 20
- Time
  - SOIL::Time, 128
- time
  - SOIL::Figure< T, x, y >, 75
- to\_json
  - SOIL, 23
- to\_json< double >
  - SOIL, 24
- to\_json< int >

- SOIL, [24](#)
- to\_json< int64\_t >
  - SOIL, [24](#)
- to\_json< SOIL::BOOL >
  - SOIL, [24](#)
- to\_json< SOIL::ENUM >
  - SOIL, [24](#)
- to\_json< SOIL::TIME >
  - SOIL, [24](#)
- to\_json< std::string >
  - SOIL, [25](#)
- to\_value
  - SOIL, [25](#)
- to\_value< bool >
  - SOIL, [25](#)
- to\_value< double >
  - SOIL, [25](#)
- to\_value< int >
  - SOIL, [26](#)
- to\_value< int64\_t >
  - SOIL, [26](#)
- to\_value< SOIL::ENUM >
  - SOIL, [26](#)
- to\_value< SOIL::TIME >
  - SOIL, [26](#)
- to\_value< std::string >
  - SOIL, [26](#)
- topic
  - MQTT::MessageContainer, [89](#)
- UDP, [27](#)
- UDP::Broadcast, [29](#)
  - ~Broadcast, [30](#)
  - add\_client, [30](#)
  - Broadcast, [30](#)
  - configure, [31](#)
  - remove\_client, [31](#)
  - send, [31](#)
  - use\_count, [32](#)
- UDP::Configuration, [38](#)
  - ~Configuration, [39](#)
  - clients, [40](#)
  - Configuration, [39](#)
  - iterator, [39](#)
- UDP::Exception, [68](#)
  - ~Exception, [68](#)
  - Exception, [68](#)
- unit
  - SOIL::Figure< T, x, y >, [76](#)
- update
  - SOIL::Figure< T, x, y >, [74](#)
  - SOIL::Variable< T, x, y >, [141](#)
- uri
  - MQTT::Configuration, [34](#)
- use\_count
  - UDP::Broadcast, [32](#)
- username
  - MQTT::Configuration, [37](#)
- utc\_now
  - SOIL::Time, [131](#)
- uuid
  - SOIL::Element, [61](#)
- value
  - SOIL::Figure< T, x, y >, [76](#)
- Variable
  - SOIL::Variable< T, x, y >, [134](#)
- verify
  - MQTT::Configuration, [37](#)
- websocket
  - MQTT::Configuration, [37](#)
- wjson
  - SOIL::Container< T, -1, -1 >, [48](#)
  - SOIL::Container< T, x, -1 >, [51](#)
  - SOIL::Container< T, x, y >, [45](#)
  - SOIL::Element, [59](#)
  - SOIL::Figure< T, x, y >, [75](#)
  - SOIL::Function, [84](#)
  - SOIL::Object, [95](#)
  - SOIL::Parameter< T, x, y >, [101](#)
  - SOIL::Range< ENUM >, [112](#)
  - SOIL::Range< std::string >, [114](#)
  - SOIL::Range< T >, [110](#)
  - SOIL::Variable< T, x, y >, [142](#)
- write
  - SOIL::Figure< T, x, y >, [75](#)
  - SOIL::Parameter< T, x, y >, [102](#)
  - SOIL::Variable< T, x, y >, [142](#)